2006

Multi-paradigm frameworks for scalable intrusion detection

Benjamin David Uphoff
Iowa State University

Multi-paradigm frameworks for scalable intrusion detection

by

Benjamin David Uphoff

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Johnny S.K. Wong, Major Professor
Thomas Daniels
Shashi Gadia
Ying Cai
Samik Basu

Iowa State University

Ames, Iowa

2006

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Benjamin David Uphoff

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

TABLE OF CONTENTS

List of Figures

List of Tables

Acknowledgments

Many people have made this dissertation possible but I would like to mention a few people without who's support I could not have been successful as a graduate student. First I would like to thank Dr. Johnny Wong for his patience and support of my often-erratic work and school schedule. Also I would like to thank my committee members (Dr. Tom Daniels, Dr. Sashi Gaida, Dr. Samik Basu and Dr. Ying Cai) for their involvement with my research.

At LANL I would like to thank my managers over the years for their support and patience: Steve Tenbrink, Ron Wilkens, Alex Kent, Dale Land and Gerry Graham. Also I would like to thank my team members for all the ideas, thoughts and suggestions. Specifically I would like to acknowledge the contributions of Susan Coulter, Paul Criscuolo, Alex Brugh, Eugene Gavrilov, Mike Fisk and Neale Pickett.

My family and friends have been critical to my success as well and I would like to acknowledge their contributions especially Laura and David Uphoff, Jared Uphoff and Rebecca Boelk. I would also like to thank Lars and Jean Brudvig for their amazing friendship these past four years in Ames.

So many people in Los Alamos have supported me the past five years but I would like to give a special thanks to the Idler family (Craig, Valerie, Alissa and Garrett) for being my "adopted" family in Los Alamos. Thanks as well to Carmella Nogar (and her family too) for her friendship, kindness, caring and all the wonderful memories.

Lastly I would like to thank Dr. David DeWitt at UW-Madison who got me started in computer science and has always been one of my greatest supporters

1. Introduction

Three principals define the goals of Information Security: confidentiality, integrity and availability. *Confidentiality* assures that data is only accessible by those persons or processes with proper authority. *Integrity* assures that data cannot be modified and that its authenticity is verifiable. *Availability* assures that users can access data as needed and that external forces cannot prevent their access. Only when these three conditions are guaranteed can an information system be considered secure.

The study of computer and network security involves applying the principals of Information Security to computer systems and networks. Modern operating systems provide a variety of mechanisms to achieve security of individual computers including access controls and host-based firewalls. Networks can likewise be protected from external attacks by deploying firewall, authentication and encryption systems.

Unfortunately, development of computer software is prone to human error, introducing vulnerabilities into software and making it impossible to guarantee confidentiality, integrity and availability in practice. On many occasions, these errors are found in the very systems responsible for protecting the computer or network. These factors have led to a proliferation of computer viruses, worms and malware, all of which lead to insecure computer and network environments. Anti-virus and anti-malware software provide a partial solution to the problem by stopping known attacks from affecting computer systems. However, this software must be constantly updated and is not deployed on every computer in practice.

Although human error in software development is an important contributor to information security failures, even correct software cannot solve problems caused by malicious insiders and incorrectly configured software. A malicious insider is a legitimate user that uses his or her privileges to violate security policies making detection difficult. This threat, commonly called the *insider threat*, is difficult to stop as the user may have organizational knowledge of security policies. Incorrectly configured software is a difficult issue facing computer and network security. A firewall, no matter how advanced, does not secure the network if it is not configured in a manner such that it adequately defends the resources it is intended to protect.

Given the prevalence in software errors, the threat of malicious insiders and the possibility of incorrectly configured software, there will always be a need to detect failures in security policy. The field of intrusion detection provides techniques and algorithms for discovering activity violating the security policy of the computer or network.

1.1. Intrusion Detection Systems

Research and development of intrusion detection systems (IDSs) gained interest in the early 1990s as the Internet gained in popularity and computers and networks became more interconnected. Where in the past home computers and local area networks (LANs) were the most common usages of computers and networks, the prevalence of the Internet shifted the computing industry towards a highly connected topology. The drawback of this connectivity is that home computers and LANs are no longer protected by way of their isolation, and are therefore more prone to external threats. For this reason, the need for detection tools continues to grow in consumer, business and government markets.

The Internet era has driven the need for IDSs and numerous research projects, open source projects and commercial products exist to fill this need. A common element of IDS designs is the use of alerts to notify the user, typically a security analyst, that malicious activity has occurred or is in progress. Alerts are typically viewed via a console application although mechanisms exist to send email and pager notifications.

IDS accuracy is measured in terms of *false positive* and *false negative* rates. A false positive is an alert generated for an event that is not an intrusion. A false negative is the failure of the IDS to detect a legitimate instance of malicious activity. These two factors are inversely related. False positives rates decrease as the sensitivity of the IDS is reduced. This causes the false negative rate to increase, as the IDS is no longer detecting as many intrusions.

IDSs fall into a number of different categories, or *paradigms*. A *host-based* IDS is executed locally on the host it is monitoring. A *network-based* IDS typically monitors a network segment by passively monitoring activity on the network. In addition to being categorized as host-based or network-based, IDSs can be *anomaly detecting* , *misuse detecting*, or *specification-based* systems. Anomaly detectors define a normal profile of

acceptable behavior and then generate alerts when activity contrary to the profile is detected. Misuse detection systems, also called *signature-based* systems, describe what constitutes malicious activity, typically as a set of attack signatures. If activity is detected that matches the signature then an alert is generated. Specification-based systems formally define the behavior of a process or application. Any action taken that does not meet the specification is considered malicious and results in an alert being generated.

The advantage of anomaly detection is that it has the ability to detect unknown attacks. Misuse detection systems can precisely identify the activity that generated the alert and traditionally have low false positive rates. As with anomaly detectors, specification-based systems can detect unknown attacks and typically have a lower false positive rate than anomaly detectors. Hybrid systems have been designed to combine the strengths of the various paradigms, although these systems are not widely deployed in practice.

For the purposes of this dissertation, only network-based systems are considered, although our techniques can easily be applied to host-based systems.

1.1.1. Practical application

In production environments, security analysts monitor IDSs to identify malicious activities on the network. An IDS instance can generate hundreds or even thousands of alerts per day depending on the type of IDS and the sensitivity of its detection algorithm. Additionally, organizations typically deploy more than one IDS, be it multiple instances of the same system or other types of IDSs. This presents a challenge to the analysts responsible for monitoring the network as the alert volume can easily surpass their analysis capabilities.

Additionally, the volume of information generated by IDSs and network monitoring tools presents a challenge in term of storage and access. Large networks can produce hundreds of millions of log records per day with storage overhead of gigabytes per day. IDS systems typically rely on a periodic archival of older alerts to maintain adequate performance of the backend storage system, typically a relational database or a set of flat file logs. This limits the analyst's ability to identify long-term trends and ties data availability to system resources.

One possible solution to the problems associated with alert volume is to reduce the sensitivity of the IDS such that it generates a low number of events. As stated previously, lowering the sensitivity of the IDS will increase the false negative rate. This is unacceptable in practice as reducing the IDS's ability to detect attacks makes analysts blind to potential attack scenarios.

Even an IDS with a low alert volume requires additional supportive information for a user to accurately determine the validity and severity of an alert. This information, called *correlated events*, comes most frequently in the form of packet data or network flow records collected by the IDS or another network monitoring device. Analysts use this information for forensics and to determine the scope of the attack. In practice, correlated events are found in a variety of different locations including flat files, relational databases and XML documents, requiring analysts to manually retrieve information from multiple, heterogeneous datasets. The process of gathering this information is called *event correlation.*

Managing the volume of alerts and supportive data is a challenge for IDS research and is directly addressed by this dissertation.

1.2. Thesis Statement

Research in network intrusion detection focuses primarily on small or artificial datasets for the design and implementation of IDSs. Tools are developed that perform well on these datasets but have failed to extend this performance to large-scale production network environments. To overcome this problem, improvements must be made to the foundations of intrusion detection systems to provide developers with the tools necessary to build IDSs that meet the needs of production networks.

This dissertation addresses four areas of intrusion detection that are important in the future development of IDS: data management and access, alert information quality, alert correlation and performance evaluation. For each of these areas, a framework design and implementation is presented to address current limitations in the scalability and usability of IDS in production environments.

1.3. Contributions

The objective of this dissertation is to provide innovative solutions to the problems identified in the thesis statement. The solutions proposed in this document represent four primary contributions to the field of intrusion detection. The first contribution is a system that allows for scalable capture and retrieval of network traffic data. A domain-specific query language is defined to allow information retrieval of heterogeneous datasets via a web service. The system is largely dependent on XML and thus can be easily extended to a variety of network environments.

The second contribution is a distributed, agent-based framework for alert verification and event correlation that addresses the need for improved quality in IDS alerts. Each IDS alert in the system is represented as a feature vector containing information regarding the validity of the alert along with correlated events that pertain to the alert. Agents deployed for alert verification and event correlation leverage the data management framework web service for scalable query capabilities.

Third, we present a multi-paradigm alert correlation algorithm that correlates alerts using feature vectors generated from correlated events. Feature vectors are then compared with a distance function and similar alerts are grouped into clusters. This approach is IDS-independent, as it does not rely on features in the alert schema or expert rules to perform correlation, making it ideal for environments running multiple types of IDS. The algorithm leverages the event correlation capabilities of the alert verification and event correlation framework in conjunction with the data access framework to retrieve correlated events for individual IDS alerts.

Fourth, we present techniques developed for tuning and evaluating the performance of alert correlation algorithms, which is applied to support the effectiveness of our multi-paradigm alert correlation technique. We use a combination of domain expert knowledge and performance metrics suitable for large, unlabeled datasets to tune results. Correlation results for a sample of alerts are evaluated by the security analysts based on their domain knowledge. Furthermore, evaluation criteria are developed to further determine the algorithm's performance. While most techniques rely on labeled audit datasets to compute the algorithm's classification accuracy, our approach evaluates correlation performance

based on any number of user-defined evaluation criteria. We then utilize the evaluation criteria to perform threshold and feature selection on unlabeled datasets.

1.3.1. Data management and access framework

The first contribution is a flexible, scalable framework for the collection and retrieval of network traffic and intrusion detection datasets. Incoming data streams are captured and indexed by the framework leveraging distributed computing for scalability. Once captured, datasets can be queried using an XML-based query language designed for the domain of intrusion detection and network traffic analysis. A single query can gather results from any number of heterogeneous sensors, IDS or log files. Furthermore, the system leverages the static, time-series nature of the datasets to provide query performance optimizations. Also, a web service application programming interface (API) is defined that allows for multi-site fusion of information for global-scale intrusion detection.

By using the web service API, analysts can access datasets outside the scope of their network by interacting with other trusted web services. This system changes the scope of the security analysts' vantage point in determining the extent of an attack. As network attacks become more complex and distributed, this functionality will be necessary for analysts to accurately assess the impact of a security incident.

This framework has been implemented at Los Alamos National Laboratory (LANL) to provide network security analysts with the ability to query billions of log records from multiple sources through a single web application. The system, Distributed Signature and Anomaly Real-time Monitoring (DiSARM), has been used in production at LANL since 2002.

1.3.2. Alert verification and event correlation framework

The second contribution is an alert evaluation framework for alert verification and event correlation. This framework can be used to validate IDS alerts to reduce the false positive rate. It is also used to automate the task of event correlation by automatically gathering events related to an alert.

In this framework, IDS alerts from multiple sensors, and potentially different paradigms, are sent to an alert broker. Distributed alert evaluation agents request alerts from the broker to perform alert verification and event correlation tasks on the alerts. Alert verification agents reply to the broker with a scalar value between -1.0 and 1.0, with negative values indicating that an alert is invalid. Positive values indicate that the alert is valid, according to the agent. A value of zero indicates an unknown or unprocessed result. These values are stored by the alert broker as a feature vector mapped to the IDS alert. The vectors can be used to rank and categorize alerts.

Agents can also be used for the task of event correlation. In this process, data and information pertaining to an alert is gathered via the data management framework's web service or other data sources such a network knowledgebase or ontology. Correlated events are stored as XML in the alert broker and can be used by alert verification agents for additional analysis. Correlated events are also useful to analysts. Without correlated events, IDS alerts, especially alerts from anomaly detectors, are difficult to analyze. Gathering related information through the use of event correlation agents makes this task easier by automating the process.

Agents in the system are autonomous but can be configured with dependencies on other agents, allowing agents to be arranged in hierarchies to provide tiered processing of IDS alerts. Agents with expensive computation can be configured to evaluate alerts after another agent meets specific conditions. This enables complex agents to be incorporated into the system without adversely impacting the performance of other agents.

1.3.3. Multi-paradigm alert correlation

The third contribution of this dissertation is a multi-paradigm approach to alert correlation. While current research in alert correlation relies on features in the alert schema or expert rules to perform correlation, our approach builds feature vectors from audit data collected using our event correlation agent framework. The correlated events are abstracted into feature vectors that represent a network traffic profile, or *fingerprint*, for the alert.

Feature vectors are compared using a distance function to establish the similarity of alerts. Alerts with similar network traffic profiles will be near one another in the high

dimensional space of the feature vector. Alerts within a threshold are placed in clusters, reducing the number of individual alerts that need to be analyzed by security analysts or post-processors.

Current approaches to alert correlation use fields in the alert schema, expert rules and knowledge bases to perform correlation. These factors require normalization of alert information, development of expert rules and maintenance of the knowledge base. By utilizing correlated events from common network traffic datasets to perform alert correlation, our design is IDS independent and does not rely on an alert ontology or network knowledge base. This makes our design suitable for production networks because of its flexibility and low maintenance requirements.

1.3.4. Performance evaluation and feature selection

The fourth contribution of this dissertation is performance evaluation and feature selection techniques for evaluation and tuning of alert correlation on large, unlabeled datasets. A common problem of adapting IDS research to production environments has been a reliance on labeled datasets to assess and tune performance. This is a problem for production networks because generating even a small, labeled dataset is an expensive, time-consuming process that is prone to human error. Furthermore it is impossible to be certain that a labeled dataset, no matter how accurate, is representative.

We use a combination of domain expert analysis and performance metrics to tune the results of our multi-paradigm alert correlation results. Cluster entropy and the number of clusters were selected as metrics for evaluating our results. We use these factors to select thresholds and perform feature selection on real-world IDS alert correlation data. Domain experts are leveraged to provide feedback of alert correlation results by evaluating a sample of correlated alerts.

By using a combination of user input and statistical measurements, we provide network security analysts and researchers with techniques for building, testing and tuning IDS and alert correlation systems in production networks. We feel this contribution provides an effective and efficient approach to adopt future and existing IDS research in practice.

1.4. Thesis Roadmap

In chapter 2, the state of current intrusion detection research is described and shortcomings and limitations of the various research areas are discussed. Given the limitations described in chapter 2, the remaining chapters address many of the fundamental limitations of current research, most importantly scalability, the quality of alerts, the number of alerts and performance evaluation.

Chapter 3 describes the web service framework for managing and accessing intrusion detection and network security datasets. Chapter 4 uses the data management framework in chapter 3 to build a framework for alert verification and event correlation. In chapter 5, the multi-paradigm alert correlation technique is described. Chapter 6 describes a feature selection algorithm that is designed to optimize alert correlation algorithms in production networks. Chapter 7 concludes the dissertation and includes a discussion of future work.

2. Related Work

This chapter reviews the current state of intrusion detection research and discusses the challenges and limitations of work in the field. Section 2.1 begins by reviewing research in various important aspects of IDS design and implementation. In section 2.2, limitations of each area presented in section 2.1 are discussed to provide motivation for the work presented in this dissertation.

2.1. IDS Research

Research in the area of intrusion detection has been very active over the last fifteen years. Early research in the field focused primarily on two types of systems: misuse detection systems and anomaly detection systems. This research established the foundation for the research that would take place from the late 1990s to the present. Today, research in the area focuses on solving the well-documented problems and limitations associated with IDS used in practice.

The focus of our research is on network-based IDSs and the following subsections examine several broad areas of past and current research in this area. Section 2.1.1 presents research in the area of network monitoring IDS, the class of IDS that is considered in this dissertation. Section 2.1.2 describes research in the area of anomaly detection. Sections 2.1.3 and 2.1.4 discuss research in the area of alert correlation by breaking the topic into two areas: alert merging and multi-step alert correlation. Section 2.1.5 discusses applications of data mining to IDS research. Lastly, section 2.1.6 describes research related to distributed IDSs.

2.1.1. Network monitoring IDS

When deploying IDSs on large networks, sensor management must be considered. Host-based solutions are often impractical, as they need to be installed and maintained on each host in the network. Therefore, network IDS are widely deployed to monitor large networks. Snort is one example of a network IDS, although it can function at the host level as well [1]. As a signature-based IDS, Snort inspects packets captured through the network

interface card (NIC), which is typically configured to passively, or *promiscuously*, capture packets. Packets are compared against signatures in a database that define what constitutes an attack against the network. Any time a packet matches a signature in the database, an alert is generated and either stored in a database or a flat file.

Another example of a network monitoring described by Sekar et al. uses a specification language to allow for the detection of low-level network attacks [2]. As a protocol verifier, the system scales over large networks but is only sensitive to a limited scope of attacks. For example, a malformed TCP packet would generate an alert, while a buffer overflow attack at the application layer would go unnoticed. This type of system is useful for detecting protocol-based attacks and can do so very efficiently due to the specialized nature of the system.

While the work developed by Sekar et al. showed that high-performance network IDS could be created, this work only allowed the monitoring of a small subset of attacks, namely protocol-based attacks. Kruegel et al. showed that a full-featured, signature-based network IDS could be created by segmenting the network data stream and performing full-fledged intrusion detection on multiple, autonomous systems [3]. These two approaches show that it is possible to monitor high-bandwidth networks without sacrificing detection performance.

2.1.2. Anomaly detectors

Anomaly detecting IDS have the useful ability to detect unknown attacks. In the case of misuse detection systems, like Snort, there exists a definition of exactly what constitutes an attack. Anomaly detectors differ in that they operate by learning the normal behavior of the system or network being monitored. Any activity that does not fit this profile is considered an anomaly, and thus suspicious.

The study of anomaly detection involves two primary phases: generation of a normal profile and then monitoring for deviations from this profile above some threshold. These basic techniques can be applied to a variety of data sets including host audit logs and network connection records. Some of the earliest and most important work in this area came out of the University of New Mexico in the late 90s. Somayaji, Hofmeyr, Kosoresow and Forrest proposed a host-based system using host system call logs that used paradigm of a computer

immune system [4, 5]. This approach served as the basis for the anomaly detector *stide*. The principle in *stide* was that a given host exhibits known and predictable patterns of behavior. After a behavioral profile is generated, the system monitors for system call sequences that do not fit the pattern. Any unusual sequence of system calls is considered anomalous and results in the generation of an alert. A variety of other extended techniques for processing system audit logs have been attempted using various techniques including Markov models, estimated weighted moving averages, finite automata and state machines [6,7,8,9].

The techniques discussed above work on log data from hosts and thus do not adapt well to a network-based IDS model. To achieve anomaly detection on network feeds, various statistical data mining approaches have been proposed. Lane and Brodley used statistical clustering techniques to establish a normal pattern of network activities using network connection records [10]. Any record that could not be clustered within an established normal cluster was considered intrusive. One drawback of this system was the need to pre-configure the anomaly detector in a training phase. The NATE anomaly detector overcomes this problem by using abnormality thresholds and can function without a training phase [11,12]. It uses clustering techniques as well, but functions at the packet header level, only allowing the detection of protocol-based exploits.

2.1.3. Alert merging

For the purposes of this dissertation, we will differentiate between techniques for alert merging and multi-step alert correlation, both of which are varieties of alert correlation. We consider alert merging techniques to be those that take multiple similar alerts and combine them to form a single abstract alert. This contrasts with multi-step alert correlation techniques, which correlate alerts through the use of modeling languages or expert rules. The commonality in these two techniques is that they combine multiple low-level, or elementary, alerts into abstract meta-alerts that are viewed by analysts.

Alert merging is useful for reducing the number of similar alerts generated by one or more IDSs. Consider the following scenario: a host on the Internet is infected with the latest worm. It begins scanning the network, attempting to spread. Assume that a signature IDS is deployed with a rule to detect the worm in the signature database. Each attempt by the worm

to spread into the network from the infected host will generate an alert. If the target network is a class B network segment, this could potentially result in 65,536 alerts, one for each IP address in the class B range, generated by a single attacking host. Of course in the case of an Internet worm, potentially thousands of hosts can attack the network simultaneously, generating millions of alerts in a short period of time.

Alert merging techniques aim to avoid the above scenario by combining multiple similar alerts into a single meta-alert. In the given example for instance, the alerts from each infected host could be combined into a single alert showing the type of attack and the number of occurrences, greatly reducing the number of alerts while still providing enough information for an analyst to determine the nature and scope of the attack.

Cuppens presented some of the most interesting work in this area [13]. This work used similarity functions based on expert knowledge to group and then merge related "elementary alerts" to create a single "global alert". Three primary dimensions for merging were presented: alert classification, time and source/target pairings. With these three dimensions, it was shown that alert reduction was effective, although the data set presented was very small, consisting of only 87 elementary alerts.

Alert representation is a related area to alert merging, as it is difficult to merge alerts in heterogeneous formats. The IETF has an ongoing effort to develop a standard, XML representation of IDS alerts [14]. This effort, the Intrusion Detection Message Exchange Format (IDMEF), makes it much easier for IDS systems to exchange alert information, although some obstacles still remain. Vendors have been reluctant to adopt the standard and there is still a possibility for data representation problems. For example, different IDS may label or categorize alerts in a different manner, making data exchange difficult. At this time, alert representation remains a problem for the intrusion detection research field.

2.1.4. Multi-step alert correlation

The goal of multi-step alert correlation is to combine individual or merged alerts instances into the various steps of a complex intrusion. As an example, many attacks start with a scan for vulnerable hosts followed by an exploit attempt using the knowledge from the scan. These two related attacks are steps in a potentially larger, more complicated attack

scenario. By linking these alerts together, alert correlation techniques provide an analyst with a deeper understanding of the larger attack scope.

Correlation techniques typically use expert knowledge to establish how alerts are correlated. Valdes and Skinner use a probability matrix to determine how incoming alerts should be correlated within the EMERALD system [15]. This technique has shown to be effective on the 2000 DARPA Lincoln Labs data set although the accuracy of the probability matrix was essential for adequate performance. Another system used the JIGSAW predicate language to chain together alerts [16]. This system requires a great deal of domain knowledge to be effective. Additionally it leans heavily on network facts such as host information to provide effective correlation. A similar work by Cuppens and Miège used a similar specification language, LAMBDA, to achieve correlation [17].

2.1.5. Data mining

A great deal of research in intrusion detection has involved the application of data mining techniques to the domain of intrusion detection, some of the most interesting research by Lee and others can be found in [18] and [19]. Data mining techniques are typically used for constructing anomaly detectors as shown in [10], [11] and [12]. The typical life cycle of an anomaly detector using data mining consists of four phases: train, test, run and revise.

First, a data mining algorithm is applied to a set of training data. Training data can be labeled or unlabeled. Techniques that use labeled data are called supervised algorithms, while those using unlabeled data are called unsupervised algorithms. Obviously, the benefit of unsupervised algorithms is that building the training data set is much easier. Labeling each record in a data set for supervised techniques can be an extremely cumbersome process, however the accuracy of systems using labeled data is typically higher than systems utilizing unsupervised techniques.

The testing phase is used to determine if the resulting model generated by the algorithm are accurate. A separate test data set is run through the system and the results are evaluated. If the results are acceptable, the system can be brought online to detect intrusions.

After entering the run phase, the anomaly detector will function as any IDS, sending out alerts when malicious activity is detected. The benefit again is that these systems can

detect unknown attacks, as they detect deviations from a normal profile generated in the training phase. At some point the system's performance is likely to degrade as the normal profile of the system shifts over time. The system must be brought offline for evaluation and then retrained before being brought back online.

Techniques exist that are capable of avoiding the training phase altogether, an approach known as *unsupervised learning*, and adapt to change in system behavior over time [11,12,20]. Yang et al. show an efficient use of clustering techniques in such a system. Skipping the training phase is obviously a powerful feature, although such systems have been shown to be less accurate than comparable techniques.

2.1.6. Distributed IDS

In large networks it is a common practice to deploy multiple IDS to monitor the network. This has led researchers to address techniques for cooperation and collaboration between various IDSs. One effort that received considerable attention and funding was the Common Intrusion Detection Framework (CIDF) [21]. This research took place in the late 90s but has since been largely abandoned. The goal of this work was to define protocols for heterogeneous IDS to function in a distributed environment. Through this system and extension projects [22], the framework allowed IDS to share alerts and data streams in an attempt to provide analysts with a better understanding of distributed attacks against the network.

Many efforts exist to implement distributed IDS systems using software agents. The Modile Agent Intrusion Detection System is one such system developed by Helmer et al [22]. This system inspired the alert verification and alert correlation framework that is presented in chapter 4 of this dissertation. Our goal was to utilize a similar, although less complex, agent framework that focuses only on the tasks of alert verification and event correlation, which are the crucial components of our approach to alert correlation described in chapter 5.

2.2. Limitations of Current Research

The research detailed in the previous section provides solutions to many of the problems facing intrusion detection. Merging algorithms and correlation techniques can be applied to reduce the number of elementary alerts that an analyst needs to address. Network monitoring systems can monitor high bandwidth networks without failure. Anomaly detectors can be deployed to detect unknown attacks. However, these ideas all fail in at least one of the two areas when implemented in large network environments: data storage and data access. Data storage is the task of collecting data items gathered by the IDS. At the most basic level this could be packet capture or system logs. Data access is the task of retrieving data relevant to the alerts that the IDS generated. For instance, a Snort alert is of little use to a security analyst if there is no supporting data stored by the sensor.

Perhaps most importantly, is the discussion of false positives. IDS research is often criticized for the number of false alarms that systems raise. The problem of false positives is much worse in large networks with multiple IDSs. If false positive rates cannot be improved upon by IDS research and applied in practice, IDS will continue to receive criticism.

The following subsections address key research areas in intrusion detection and show where the failures lie and why they are of importance to the overall security of the network. The goal of the following subsections is to motivate the need for the frameworks and solutions presented in chapters 3 through 6.

2.2.1. Network monitoring limitations

An important concern of any network monitoring IDS is how to handle vast streams of incoming data. Packet analysis systems like Snort have little choice but to throw away the vast majority of the data stream. Snort can be configured to store the payload of relevant packets within the alert records. Even this seemingly small subset can quickly degrade the tool's performance and consume valuable disk resources. At a minimum the data that triggered the alert must be stored so that analysts can verify the alert.

Ideally, IDSs attempt to store as much data as possible although this presents many challenges to IDS design and implementation. A packet-based monitoring solution can only keep a small window of packet data before the storage overhead becomes prohibitive. In

large networks, packet data rates are on the order of hundreds of gigabytes per day. It is unrealistic to expect to keep more than a few days of data online at any given time. Even then, accessing the data in an efficient manner is still very problematic, not to mention the hardware costs for such a system.

To avoid keeping packet data, network connection summaries can be assembled from raw packets. This approach has several advantages in terms of data reduction. A session assembly algorithm can bring the data set size down from the hundreds of gigabytes of packet data to one or two gigabytes of network connection records. This is still a large amount of data, but it provides analysts with a cleaner, yet still information-rich, data set.

Even after applying network connection assembly techniques, data access is still problematic. A high-bandwidth network can easily gather one terabyte of connection records over the course of a year. Chapter 3 provides solutions for applying data access techniques to these types of large datasets.

False alarms are another problem common to network monitoring IDSs. To minimize false positive rates, analysts must continually tune signatures to better match the attack patterns. This requires significant time and expert domain knowledge on the part of the security analyst.

There are some cases however, in which no amount of tuning can eliminate false positives. Take for example a rule monitoring for a buffer overflow attack on a web server that alerts on a sequence of 100 or more null characters. In the case of a buffer overflow, the null characters are used to pad the buffer to overflow the stack and inject malicious code. Although this seems like a robust rule when tied to a specific port, in this case port 80 for the web server, this alert has a tendency to raise an alarm on certain JPEG images that by chance contain this sequence. The analyst has no choice but to accept the possibility of false positives or remove the rule, leading to false negatives where real instances of the buffer overflow go unnoticed.

2.2.2. Anomaly detector limitations

As alerting mechanisms, anomaly detection systems (ADSs) raise alarms anytime the behavior of the network or host system exhibits unusual behavior. Typically, these alerts

report only that something abnormal has occurred for the host or network being monitored. This also allows anomaly detectors to detect unknown attacks. While the ability to detect unknown attacks and changes in system behavior is an important trait of ADSs, alerting that an anomaly occurred is not sufficient to provide an analyst with enough information to react to a potential attack. The following sections make similar arguments for the areas of alert merging and multi-step alert correlation. The commonality of these cases is that alerts need supporting information to be useful to end-users. For anomaly detection, an alert on a previously unknown attack would be confusing to the user without supporting data. However, with access to the related information that triggered the alert, the user can make an informed decision about what was actually occurring in the network or on the host to generate the anomalous event.

To provide this level of access to the data, a supporting infrastructure must exist. The previous section argues that data access is a problem for network-based IDS. Similarly, a host-based anomaly detector can generate tens of megabytes of system logs per day, depending on its configuration. Thus, a large network of 1000 hosts running the same anomaly detector would generate system logs in the ten-gigabyte per day range.

Although false positive reduction is an active area of anomaly detection research, no system to date has been shown to function reliably in practice with a low false positive rate while maintaining a high detection rate. The nature of these systems indicates that false positives will always be an issue. In general, a higher detection rate for the anomaly detector will result in a higher false positive rate. Supportive mechanisms must be developed to help analysts reliably determine when an alert is a false positive and when something truly unusual is taking place.

2.2.3. Alert merging limitations

Research in alert merging provides a means of reducing the number of overall alerts to be evaluated by security analysts. As with other areas of research, one problem with these techniques is that data access is largely ignored. Merging algorithms group together alerts in ways that can be difficult for the analyst to understand. In an operational setting, the analyst would have to verify that the aggregated alert was properly grouping alerts together before

acting on a particular incident. In some cases this would require more than simply breaking apart the alert and looking at each elementary alert; the actual data would need to be analyzed.

There are two possible solutions to this problem. The first and most apparent would be to store the corresponding data with the alert. Snort does this by saving the offending packet, when applicable, at the user's discretion. This is an expensive procedure and causes a data explosion problem in the alert database. The problem with this method is evident in the case of anomaly detectors. Consider the alert "system A attempted connections to 1000 hosts in 15 minutes". This alert would be indicative of scan activity, which is considered malicious behavior in most cases. However, when it is time to store the alert it is unclear exactly what data to store along with the alert. Clearly, storing 1000 data items, the connection attempts, would be the most useful information, however, the storage costs would quickly become prohibitive, given the frequency of scan activity in the Internet.

The second approach would be to store all the logs separately from the alert and allow for a mechanism to retrieve those logs that correspond to the alert. This is a more scalable solution, as the alert database will not suffer from data explosion. Certainly, the cost of storing large volumes of log data will be very high. One goal of this dissertation is to argue for the importance of storing and allowing fast access to large volumes of log data with minimal overhead. Providing analysts with a deeper understanding of why an alert was triggered is the first of many justifications for the storage of log data.

Although these techniques help to reduce the overall number of alerts that an analyst must evaluate, they do nothing to address the number of false positives. If a meta-alert contains 1000 elementary alerts and 99% of them are false positives then little is gained by merging the alerts, as the analyst still must evaluate each alert to determine whether it is valid or a false positive.

Alert merging solutions rely on a normalization phase prior to correlation in which alerts from heterogeneous sensors are converted into a uniform alert schema. Often this schema is used to correlate alerts by merging alerts on fields in the schema, limiting these approaches to systems that can be normalized.

2.2.4. Alert correlation limitations

In alert correlation, a number of alerts represent steps in a single, multi-step attack, sometimes called a "meta" or "global" alert. This differs from a merged alert, which contains multiple occurrences of the same, or similar, activity. It is also worth noting that correlated alerts could be placed in clusters, as they should act like just another alert in the IDS. Although somewhat different, correlated alerts suffer from the same problem inherent in alert merging. The data items that triggered each sub-alert must be accessible or the analyst will not be able to verify the correctness of the meta-alert. Even if the correctness of the meta-alert can be verified, it can still be difficult to understand the meaning of the alert as multi-step attacks can be confusing, even for the most experienced analyst.

Furthermore, the techniques described in section 2.1.4 all require a great deal of domain knowledge to be effective. In many cases this knowledge is not readily available, at least to the degree required by these systems to be useful. Also this domain knowledge must be translated into either a probability matrix or a specification language (e.g. LAMBDA or JIGSAW) [15,16,17]. This is a non-trivial problem and requires a great deal of interaction and maintenance on the part of security analysts. Failure to maintain the knowledge base will results in uncorrelated or miscorrelated alerts, resulting in more work for the analyst in the long run.

Lastly, all the systems discussed here are bound by time constraints. A knowledgeable attacker will take advantage of such systems by moving slowly enough as to slip outside of the time bounds. As these hackers are the most dangerous due to their patience, these systems can provide a false sense of security by not correlating events outside of the time bounds. In chapter 5 we present an alert correlation approach that does not temporally bind the correlation process.

It is apparent that the validity of elementary alerts that make up the steps in a multi-step alert is crucial to the overall accuracy of these systems. The attack sequence is only of interest to an analyst if all the alerts in the sequence can be shown to be accurate. A few false alarms in the chain of events could make a seemingly important global alert useless. Chapter

4 addresses this limitation by describing an agent-based framework with alert verification capabilities.

2.2.5. Data mining limitations

As described in section 2.1.5, there is considerable interest in intrusion detection-related data mining research. Research in the area of data mining, as related to intrusion detection, typically focuses on applying data mining algorithms from other disciplines to solve problems inherent to IDS. There is often little discussion of implementation, as is the case in [18], making it difficult for software developers to build these systems in practice.

Most notably overlooked is the cleaning, or *normalization*, phase of data mining. This is typically the step that follows data collection in the data mining process. Most intrusion detection experts are not data mining experts. The typical analyst lacks the expertise required to clean and prepare data for this phase. If an infrastructure exists in which analysts can automate the critical phases of collection, cleaning and preparation, the implementation of data mining techniques becomes feasible for intrusion detection experts. Our work described in chapters 3 and 4 directly addresses this issue.

Almost all anomaly detection research involves tuning the system to maximize the detection rate while minimizing the false positive rate. Most of this tuning takes place using the DARPA Lincoln Labs data set, which is labeled [24]. This makes it easy for researchers to find the optimal settling for the algorithms used to detect intrusions. In practice however this is a non-trivial problem as it is impossible to perform this type of tuning on a live network where the nature of the activity is not labeled.

2.2.6. Distributed IDS limitations

Attempts at providing collaboration between IDS have run into problems due to the additional complexity required for effective collaboration. Complex protocols and information exchange difficulties have diminished efforts in this area. The only current topic of interest in the area is the IDMEF specification. By representing alerts in a common data format, heterogeneous IDS sensors can interpret alerts without the overhead of data exchange

protocols. Even so, discrepancies exist in alert representation even when using the IDMEF. Naming and categorization of alerts amongst different IDS is a common problem that is not easily solved.

The most extensive attempt at a framework for cooperative IDS was the CIDF project. Due to the rigid protocols and requirements of systems interoperating with each other, the IDS vendor community never adopted the project. The lesson learned from the CIDF is that a more flexible architecture needs to be developed before cooperative, distributed IDS systems can be widely implemented. Such a framework is described in this dissertation, both in terms of alert and data management, and addresses the issues that caused the failure of the CIDF.

3. Web Services for Network Security Datasets

This chapter describes a web service framework for multi-site data sharing of heterogeneous network security datasets, a basic building block for large-scale distributed intrusion detection. The concepts and results presented in the remaining chapters rely on this framework for data access to large heterogeneous datasets, making it the foundation of this dissertation.

An interface is defined that allows single query access to multiple data sources including network logs and intrusion detection system alerts. Queries are submitted as extensible markup language (XML) documents and results are returned as XML documents via a single web service operation. A prototype system is deployed at Los Alamos National Laboratory (LANL) and is used extensively for network security operations on multi-terabyte datasets including intrusion detection, event correlation and network forensics.

3.1. Introduction

Access to network security data is critical to the tasks performed by network security analysts in protecting their networks. Law enforcement officials also require access to this data when performing investigations for cyber crimes. Additionally, access to network security data is essential to homeland security initiatives and critical infrastructure protection. The trend towards complex distributed attacks on computer networks makes this task difficult, forcing cooperation amongst sites to determine the larger attack scope. It is no longer sufficient to simply monitor one network segment. Relevant data must be obtained from distributed sensors both internal and external to the target network. We address the problems inherent in multi-sensor data fusion [25] by defining a framework that utilizes web services to allow for real-time, multi-site data fusion for network traffic and intrusion detection datasets. The system also provides mechanisms for scalable management of large datasets common to network security operations.

The design of this framework was driven by the requirements of security analysts and the nature of the types of data used in their daily operations. When performing an

investigation of a possible intrusion, analysts need a means to quickly access related records from multiple data sources. In many cases this is not easily done as data can reside on multiple database servers, often of differing types, and in many cases flat file logs. This type of data is usually very low level, unorganized and can be large - on the order of a terabyte or more. Additionally, there is no common format for network security data although typically there is commonality amongst datasets (e.g. IP addresses, timestamps and ports).

Some efforts, most notably the intrusion detection message exchange format (IDMEF) [14] and common intrusion detection framework (CIDF) [21, 26], have attempted to unify data sources in a common representation, however these efforts have not been widely adopted by the security developer community. In the case of the IDMEF, it is a developing standard under the Internet Engineering Task Force (IEFT). As the IEFT gets closer to completing the specification, more tools will hopefully adopt the standard. Currently, a library exists for the popular intrusion detection system Snort that allows alerts to be outputted as IDMEF messages. The CIDF's goal was to enable interoperability amongst intrusion detection systems. Unfortunately, this effort is no longer under development and has never been adopted by the industry.

Given the needs of analysts and the heterogeneous nature of the datasets involved, a web service using extensible markup language (XML) to represent query requests and results was developed. The service accepts queries represented as XML documents and returns results to the client in the form of an XML document. All communications are made via Hyper-text Transfer Protocol (HTTP) allowing the system to be easily deployed across multiple sites and secured using the site's existing firewall and Secure Socket Layer (SSL) infrastructure. Furthermore, each site designs and implements its own result schema by extending a generic document type definition, an important aspect as sites are unlikely to have the same set of tools and sensors for intrusion detection and network monitoring.

There are three contributions in this chapter. First, we define the basic network security web service API, which can be further extended to meet the needs of the site deploying the service. Second, we define a declarative XML-based query language and its accompanying result XML schema used by the web service for returning results as XML documents. Finally, we describe and evaluate the prototype design and implementation, named

Distributed Signature and Anomaly Real-time Monitoring (DiSARM), deployed at LANL. DiSARM provides security analysts with single query access to multiple network security-related datasets that comprise multiple terabytes of online data.

The work described in this chapter describes a design and implementation of our prototype system that shares many characteristics of a database mediator [27], using an XML query language as the common view across the various data sources. Mediation systems are differentiated from data warehouses in the database research community in that they do not store any data. Instead, data is stored in already existing storage, such as relational database servers. The data is then pulled from these sources when a query is submitted to the mediator, in this case the web service. Our system differs from traditional mediator systems in that our prototype captures and provides access to multi-terabyte network security datasets in real-time, in addition to providing access to data already stored in relation databases and documents. As it stores data in some instances, the system can be viewed as a hybrid mediator and data warehouse system.

Applying the web service concept to network security and intrusion detection provides analysts a new means of gaining perspective for network defense. Sites can utilize this framework to cooperate by sharing information to detect complex distributed attacks. By deploying a web service instance, sites can help the larger homeland security initiative by making their logs available to law enforcement officers or other government agencies. The need for this service will become more important if legislation similar to the U.K.'s Anti-Terrorism, Crime and Security Act are adopted in other parts of the world [28].

3.2. Web Service Overview

At the core of any distributed computing system is the messaging framework. Remote procedure calls (RPC) and remote method invocation (RMI) are such examples, enabling hosts to invoke function and procedure calls on remote servers [29, 30]. Web services can be seen as an updated version of RPC built on a foundation of XML-encoded messaging. Simple Object Access Protocol (SOAP) is a popular protocol for the message passing capabilities of a web service. Web service description language (WSDL) is a specification for publishing a web service so that applications can write client applications to utilize the

service. Universal Description, Discovery and Integration (UDDI) is a specification for deploying a web service so that client applications can find and utilize the web service [31].

The extensibility of XML is a key component in web services, making it much easier to add features than traditional means like RPC. For example, adding a parameter to a function call in RPC would require both the client and server to recompile their applications. By using web services the feature is simply published by the web service host using WSDL. The client can immediately make use of this new parameter through the SOAP protocol by including the information in the SOAP XML message. This is especially important in the Internet, where potentially millions of clients could be using a dynamic service.

The concept of web services has seen considerable interest in various industries including banking, stock markets and media. An example of web services is the API provided by Google that allows developers to write software that can submit searches to the Google search engine and receive results as structured data [32]. Using the web service, software can be written to integrate Google search capabilities in ways that would not be possible without a means of gathering search results as structured data.

Given the pervasiveness of relational databases as a storage medium, it must be discussed why an XML web service approach to multi site data sharing is superior to simply using an Open DataBase Connectivity (ODBC), or Jave DataBase Connectivity (JDBC), infrastructure for sharing sensitive network security data sources. The approach described herein provides a uniform language and consistent mechanism for querying and retrieving data using XML and HTTP. Even if we assume that data resides in relational databases, there are SQL syntax and implementation differences amongst database implementations. We may not be able to assume that the supported syntax will be the same across different servers. For example, the MySQL timestamp "2005-7-4 12:00:00" is represented in Oracle as "4-JUL-2005 12:00:00 PM". Although ODBC and JDBC provide some abstraction to alleviate these problems, users will at some point write ad-hoc queries and need to be prepared for the various syntax differences between systems.

While syntax issues amongst servers is a big problem to overcome, querying multiple data sources is cumbersome and difficult to automate if data resides on multiple database servers of differing types. This also becomes a security issue when sharing data across

multiple sites. The default ports for Microsoft SQLServer, Oracle, MySQL and PostgreSQL are commonly scanned ports on the Internet due to the numerous vulnerabilities associated with each product. Exposing database servers to the Internet is a risky approach to sharing network security data between sites. Using a web service framework alleviates many of these problems by functioning over HTTP and utilizing SSL and firewall rules to safely share data between sites.

Authentication and access controls are fundamental aspects of the web service framework described in this chapter. All widely used commercial and open source database servers implement user authentication in some manner. However, as each server implements these features differently, it is difficult, if not impossible, to provide uniform protection to datasets stored in different servers. This becomes both a management and security risk in setting up such a system. In our current design and implementation, users gain the benefits of single-sign on via the web service while at the same time allowing for mandatory access controls on a per-record level of granularity.

3.3. Network Security Web Services

The web service infrastructure relies on two XML schemas, query and result, to provide the abstraction necessary to enable multi-site sharing of network security data. The query schema allows for users to perform a single operation to request data from multiple heterogeneous data sources. Since the language is specific to the domain of network security, analysts can write queries leveraging their expertise, without knowledge of the underlying data formats and storage architectures. This is critical for sharing data amongst sites, as it is unlikely that remote users will have knowledge of the data available at a given site. Sub section 3.3.3 details the query document schema, which is presented as a document type definition (DTD) in appendix A.

As the server composes the results for a query, they are returned to the client in an XML format published by each site. A base result schema is described in section 3.4 and presented as a DTD in appendix B but is presented as a general schema, allowing for sites to extend the result schema to fit the needs of the organization. This is important in that no two sites are likely to have exactly the same datasets or result formats available for intrusion

detection tasks and therefore will design differing result schemas. The web service interface provides a mechanism for querying the result DTD from the server, allowing the client to validate and process the result stream.

The server hosting the web service is responsible for parsing the XML query, retrieving results from the various datasets and converting them into XML if needed. Most sites store data in a variety of formats including flat file logs, relational databases and, more recently, XML documents. A dedicated query processor parses the query and searches the various data formats for matching records. As results are returned to the web service data is converted into an XML document in accordance to the site's schema and returned to the client. The abstraction provided by the server is crucial in sharing network security information, allowing analysts to query data without knowledge of how, where, or in what format data is stored.

3.3.1. Web service functions

The web services mechanisms needed to implement a multi-site network security infrastructure are straightforward, consisting primarily of query-response behavior between client and server. To execute a query, the client sends an HTTP POST request to the server over containing the query represented as an XML document. The server replies with an XML document that can be downloaded by the client or processed as a stream. The response is an XML document containing either an error message or the results of the query. Additionally, the server provides functions that provide metadata information to clients. Clients can request the data sources available to query, the query language DTD supported by the server and the result format DTD so that the client can validate the result stream.

Due to the limited set of functionality provided by the server in our prototype implementation, the HTTP protocol was sufficient for implementing the web service infrastructure. A web service protocol such as SOAP could have been chosen instead of HTTP although the added complexity in developing our prototype outweighed the benefits of such a protocol. Additionally, HTTP infrastructure is well understood by developers and can be easily secured by using SSL and firewalls.

The set of actions supported by the web service are implemented as HTTP parameters. Each HTTP request must provide an *action* parameter with one of three values: *schema*, *types* or *query*. The *schema* action returns the query DTD or the result DTD as specified by the additional *type* parameter, which contains the value *query* or *result*. The *types* action returns an XML document that defines the data sources that are made available for querying through the web service. Also included the document is information about the search criteria and field representations available for each data. Sections 3.3.3 and 3.4 explain how this information is structured and used to generate queries. The *query* action is used to submit queries to the web service and receive a result set as an XML document. When specifying the *query* action, the *query* parameter must be set, containing the XML query to execute.

3.3.2. Query language

The query language used in the web service infrastructure provides a great deal of flexibility for interacting with the web service to gather results from network traffic datasets. A single query can be used to retrieve XML data from the system matching the query criteria. Domain specific XML is used to represent queries submitted via the web service. This section provides an overview of the language and the motivations for its design. Section 3.3.3 describes the details of the language, section 3.3.4 defines rules for writing queries and section 3.3.5 provides examples queries.

We do not make claims about the power and expressiveness of the query language. The language is not intended to be a relational or Turing complete language. It is instead designed for usability by leveraging the knowledge and expertise of security analysts and developers, allowing them to retrieve data from heterogeneous datasets with a single query operation. The expressiveness of the language allows for simple selection and union operations. The language's power is built to provide the greatest common denominator among SQL, XPath, XQuery and b+ tree searches, which are used in the current implementation to index large flat file data sources. This design choice allows users to query different storage models via a consistent interface.

A domain specific language was designed to meet several needs of security analysts. First, by using keywords and constructs from the networking lexicon the language allows

analysts to leverage their knowledge of the networking domain. This differs from a general-purpose language like SQL in that elements and attributes in the XML document are familiar to those with networking backgrounds while constructs in SQL, such as SELECT and WHERE, have nothing to do with the context of the data space. The benefit of this design choice is that analysts can construct queries using their networking knowledge without understanding the details of the underlying data storage methods.

Second, a domain specific language was chosen to limit the scope of the language and tailor it specifically towards information retrieval tasks for network security data. While building the requirements for the system, it was found that analysts primarily needed simple search capabilities over multiple large data sources. A typical example would be to retrieve all records for a given IP address for the last month. Given these types of operations, the language was built to reflect this need, allowing analysts to retrieve data from multiple sources with a common language, and in many cases a single query.

Third, our design focuses on a two-phase approach to gathering and analyzing network security data. In the first phase, a subset of the data is gathered on basic search criteria, like time and IP address, obtained via information about the incident being investigated. In the second phase, the smaller dataset, in the form of an XML document, is analyzed through the use of XML processing tools like extensible stylesheet language (XSL), XPath and XQuery. The data analysis phase tends to be repeated multiple times on the same set of data. This would not be desirable to attempt on the entire dataset, as it is prohibitively large. Thus, our implementation focuses on providing a means of quickly and easily gathering data to be analyzed in the second phase.

The final design choice for the language was to allow for simple parsing and rewriting. The constructs of the language are not overly complex and can be processed using a combination of XML libraries and custom code in the query processor. Additionally, the language design is such that queries can be quickly decomposed by a variety of criteria to allow for parallel processing. For example, a query specifying three data types to search can be broken apart into three separate queries and passed on to additional query processors to compute the result in parallel.

The end result of our design is an intuitive language for querying data relevant to network security investigations. Analysts at LANL can run queries for a variety of tasks, helping them evaluate and conclude incidents much more efficiently than in the past. The following subsections describe in detail the query language specification.

3.3.3. Query language schema description

This section details the schema of the web service's query language. The complete DTD is presented in appendix A. The *Query* element can contain four elements, including additional *Query* elements nested as sub-queries. The *Timestamp* element specifies a time range to search. Including the *DataType* element allows for specific datasets to be searched. Lastly the *SearchField* element is used to specify the selection criteria for the query. Results from multiple data types and sub-queries are concatenated in the result set, performing a union.

Due to the time-series nature of the datasets used in network security, date and time information must be provided in each query document. To satisfy this requirement, one or more *Timestamp* elements must be included. The *start* attribute is mandatory and is used to specify the lower bound of the time range to search. The *end* attribute is optional and is used to specify the upper bound of the time range to search. When both *start* and *end* attributes are specified, they must be of the form "YYYY-MM-DD" or "YYYY-MM-DD HH:MM:SS.MS". The date format implies that the entire date should be searched while the timestamp format indicates the bound of the time range. If the *end* attribute is omitted, the *start* attribute must be of the form "YYYY-MM-DD", indicating that the entire date specified should be searched. Table 3.1 provides several example timestamp elements with descriptions of how the syntax is interpreted.

The usage of the optional *DataType* element specifies data sources and sensors to be queried. Omitting this element causes all data sources to be searched and included in the result set. The *type* is a mandatory identifier such as "Snort" or "Netflow" in this case specifying Snort alerts or Netflow connection records respectively. In the case where multiple sensors of the same type are collecting data, the optional *sensor* attribute can be used to specify a particular collection point. When omitted, results from all sensors for the

specified data type are returned. The values for the *type* and *sensor* attributes are obtained via the *types* action of the web service API as described in section 3.3.1.

Table 3.1. Timestamp element examples

Example	Description
<Timestamp start="2005-8-1"/>	Return records with a timestamp at any time on August 1, 2005
<Timestamp start="2005-1-1" end="2005-6-30"/>	Return records with a timestamp at any time during the first six months of 2005
<Timestamp start="2005-7-11 11:00:00.00" end="2005-7-11 13:00:00.00"/>	Return records with a timestamp between 11 AM and 1PM on July 11, 2005.

The *SearchField* element allows the specification of search criteria. The *name* attribute is the shorthand name of the field. An example name could be "srcip", representing a source IP address. The *format* attribute is the shorthand name of the data format as described in the system metadata. An example format could be "hexip", representing an IP address in hexadecimal. The *predicate* attribute defines the predicate in the search criteria. Supported predicates are greater than, less than, greater than equal to, less than equal, equal to and inclusive range. Note that the ">" and "<" symbols cannot be included in XML data and must be replaced with ">" and "<" respectively. The "~" symbol was chosen for the range predicate because it avoids confusion with negative values when used in shorthand form within an attribute in the *Query* element. The *value* attribute specifies the value to apply the predicate against when a predicate other than range is specified. If a range predicate is specified, the *lower* and *upper* attributes must include values representing the inclusive bounds of the range. Information about search fields and representations is returned via the *types* action of the web service API as described in section 3.3.1

The attributes in the *Query* element enable authentication and allow the query writer to specify search criteria in shorthand. The *user* and *password* fields are used to authenticate users trying to submit a query to the system. This mechanism can be used to implement access controls, a feature that is discussed in section 3.5.1. The remaining attributes (*srcip*, *dstip* and *dstport*) are italicized in appendix A, indicating that they are specific to the LANL implementation of the web service. Other fields can be added here depending on the search

fields available at the site deploying the web service. Values specified for the attributes are of the form "PREDICATE|VALUE" or "LOWER~UPPER", depending on the predicate chosen. If the range predicate is specified then "LOWER" is the lower bound of the range and "UPPER" is the upper bound of the range. If a predicate other than range is chosen then the predicate value ("%gt;", "%lt;", "%gt;=", "%lt;=", "=") should immediately precede the key value. If no predicate is specified, equality is assumed. As examples, *srcip="192.168.1.0~192.168.1.255"* searches for source IP addresses in the 192.168.1 subnet and *dstport="%gt;1024"* searches for destination ports greater than 1024.

3.3.4. Query syntax rules

Although most XML parsers allow for an XML document to be validated in accordance to the DTD, additional conditions must be checked to be sure that a query is correct. This section discusses these properties and establishes some rules that must be enforced by the query processor to assure that only correct queries are accepted. Assuring that queries conform to this specification is left to the query processor implementation. If these rules are not enforced, queries can be ambiguous and potentially return inconsistent results.

In addition to adhering to the DTD and the date and time requirements, some additional rules are enforced when processing queries. Table 3.2 summarizes these rules and they are explained in detail below.

Table 3.2. Query processing rules

Rule	Definition
1	Nested queries adhere to the date/time ranges specified in the parent query, unless the parent query has no recursively specified date/time range
2	Nested queries adhere to the data type specification in the parent query, unless the parent query has no recursively specified data type
3	In order to be included in a result set, a dataset must contain all search attributes specified in the *Query* element and *SearchField* elements

The first rule is that overlapping *Timestamp* elements cannot be nested in sub-queries. For example, the user cannot submit a query specifying a date range of March through April at the root *Query* element and then specify a different date range in a nested query. This rule is necessary because conflicting or overlapping time stamps can lead to ambiguity in query processing. Rule 2 is similar to rule 1, stating that nested queries cannot have conflicting *DataType* definitions that can lead to ambiguity in query processing in the same way as having conflicting timestamps.

Rules 1 and 2 refer to recursively defined time stamps and data types respectively. This language is used because nested sub queries are associated with a parent query, which may also be a nested query. This fact makes it necessary to recursively traverse a query tree in processing to identify the time stamps and data types for a query.

Lastly, rule 3 deals with mapping attributes in the *Query* element to the datasets being searched and returned in the result set. Not every dataset has the search attributes for a given query. For instance, a data source may not have a destination IP address field. When a query is requested with destination IP address as a search criterion, that dataset cannot be search and thus cannot be returned in the result set.

3.3.5. Query examples

This section shows examples of queries used to request data via the web service. The examples shown in table 3.3 illustrate several correct and incorrect queries. For brevity, the *user* and *password* attributes are omitted in the examples.

Queries 1, 2 and 3 are valid in accordance to the query language DTD and do not violate the rules from section 3.3.4. In these examples, all data sources will be queried, as no *DataType* element is specified. Query 1 performs a three-way selection match on source IP address, destination IP address and destination port for all dates in the year 2004. Query 2 has two sub queries that search for different source IP addresses over the date range defined in the parent query. Query 3 is logically equivalent to the second query but places a *Timestamp* element in each sub-query instead of the parent and also uses the *SearchField* construct instead of the *Query* attributes.

Table 3.3. Valid example queries

Query	XML
1	<Query srcip="=192.168.1.100" dstip="=1.2.3.4" dstport="1~1024"> <Timestamp start="2004-01-01" end="2004-12-31"/> </Query>
2	<Query> <Timestamp start="2005-01-01" end="2005-06-01"/> <Query srcip="=192.168.1.100"/> <Query srcip="=192.168.1.200"/> <Query/>
3	<Query> <Query> <SearchField name="srcip" predicate="=" value="192.168.1.100"/> <Timestamp start="2005-01-01" end="2005-06-01"/> </Query> <Query srcip="192.168.1.200"> <SearchField name="srcip" predicate="=" value="192.168.1.200"/> <Timestamp start="2005-01-01" end="2005-06-01"/> </Query> <Query/>

In table 3.4, examples are provided to shown incorrectly written query examples along with corrections to make the queries valid. Incorrect or ambiguous syntax is italicized in the table.

Queries 4 and 6 are incorrect according to the rules defined in section 3.3.4. Queries 5 and 7 show possible ways that the queries can be fixed to adhere to the rules. Query 4 violates rule 1, as the date ranges are ambiguous. The date ranges in bold conflict in that the parent query's range is not equal to the range in the first sub-query. This query is corrected by moving the root *Query* element's *Timstamp* element to the second sub-query, as shown in query 5. Query 6 violates rule 2 as the data types in bold are conflicting. This query can be corrected by moving the first *DataType* element into the body of the first sub-query, as shown in query 7.

Table 3.4. Invalid and corrected example queries

Query	XML
4	<Query> *<Timestamp start="2004-08-01" end="2004-09-15"/>* <Query srcip="192.168.1.100"/> *<Timestamp start="2004-09-01" end="2004-09-10"/>* </Query> <Query srcip="192.168.1.200"/> </Query>
5	<Query> <Query srcip="192.168.1.100"/> <Timestamp start="2004-09-01" end="2004-09-10"/> </Query> <Query srcip="192.168.1.200"> <Timestamp start="2004-08-01" end="2004-09-15"/> </Query> </Query>
6	<Query> *<DataType type="t1"/>* <Timestamp start="2005-06-01" end="2004-09-15"/> <Query srcip="192.168.1.100"/> <Query srcip="192.168.1.200"> *<DataType type="t2"/>* </Query> </Query>
7	<Query> <Timestamp start="2005-06-01" end="2004-09-15"/> <Query srcip="192.168.1.100"/> <DataType type="t1"/> </Query> <Query srcip="192.168.1.200"> <DataType type="t2"/> </Query> </Query>

3.4. Result Formats

With recent advances in XML technology and the heterogeneous nature of network security data, XML is a logical and effective means of representing result sets. As it is self-describing, XML can help reduce problems of ambiguity associated with this type of data. The extensible nature of XML is also very appealing when dealing with network security data as the formats and data sources are continually changing.

Commercial vendors like Microsoft and Oracle are focusing more attention on implementing XML query languages like XPath and XQuery in their relational database management systems. These emerging technologies, along with transform languages like XSLT, make XML even more appealing as a storage method for performing complex data analysis and event correlation.

It is recognized that no two sites are likely to have access to the same logs and data sources. Therefore, the result schema is generalized and is to be extended by sites implementing the web service. Omitted are the elements and attributes of the data types themselves. It is the responsibility of the site implementing the web service framework to develop, publish and maintain their own individual schema reflecting the data being made available to other sites. The base DTD is provided in appendix B.

Result documents begin with the *Result* element, which contains three possible children: *QueryError*, *DataType* and *QueryResult*. The *QueryError* element will contain an error message if there was an error while processing the query, for example a user authentication error or a malformed XML query. *DataType* elements are returned in the document when the types action is executed by the client and provide information about data types, sensors and search fields. The *QueryResult* element contains records returned according to the XML query when the query *action* is specified.

The *DataType* element is used to define the data sources that can be queried via the web service. The *id* attribute defines the unique identifier for each data source to be used in the *DataType* element of the query schema. The *name* attribute is a text name of this type that can be used in lieu of the identifier. The *DataType* element can contain as children two elements: *Sensor* and *SearchField*. Like its parent, a *Sensor* element contains an *id* and *name* attribute, either of which can be used in the *sensor* attribute of the *DataType* element in the query schema. The *SearchField* element is used to describe the fields that can be queried for the parent *DataType* via the web service.

Each *SearchField* element contains an optional *description* and a required *name* attribute. The *description* attribute provides a text description of the field while the *name* attribute is used in the *name* attribute of the *SearchField* element of the query schema when constructing queries. The *SearchField* element contains at least one child of both *Predicate*

and *Format*. The *Predicate* element has a single *value* attribute and defines the types of searches that can be executed for this field. *Format* elements contain information about data formats supported by the web service for the parent *SearchField*. The *name* attribute is the value used in the format attribute of the *SearchField* element of the query schema. The optional *description* attribute is a text description of the representation. The *default* attribute is optional and set to "yes" if the representation is that used by default by the query processor. For instance, the default format for specifying an IP address might be in the form of a dot quad IP address. In this case, the default attribute is set to "yes" for this format but set to "no" for the 32-bit integer representation of an IP address. The *format* attribute of the query schema's *SearchField* element is used to specify how the corresponding search values should be interpreted in query processing, for example <SearchField name="srcip" format="ip_hex" predicate="=" value="01020304"/> is equivalent to <SearchField name="srcip" format="ip" predicate="=" value="1.2.3.4"/>.

Lastly, the *QueryResult* element contains results returned from the query processor as an XML document. The children of this element are to be defined by the site implementing the web service. The result DTD in appendix B provides an example child element implemented in the LANL prototype.

3.5. System Design and Implementation

A prototype system, DiSARM, has been implemented at LANL to provide analysts with single query access to multiple terabytes of log data gathered since January of 2002. Datasets stored within the system include web logs, IDS alerts and network connection records. Some data, such as IDS alerts, resides in relational databases. Other, multi-terabyte datasets are stored in flat files and indexed with b+ trees on key fields, including source IP address, destination IP address, and destination port. Figure 3.1 details the prototype system architecture.

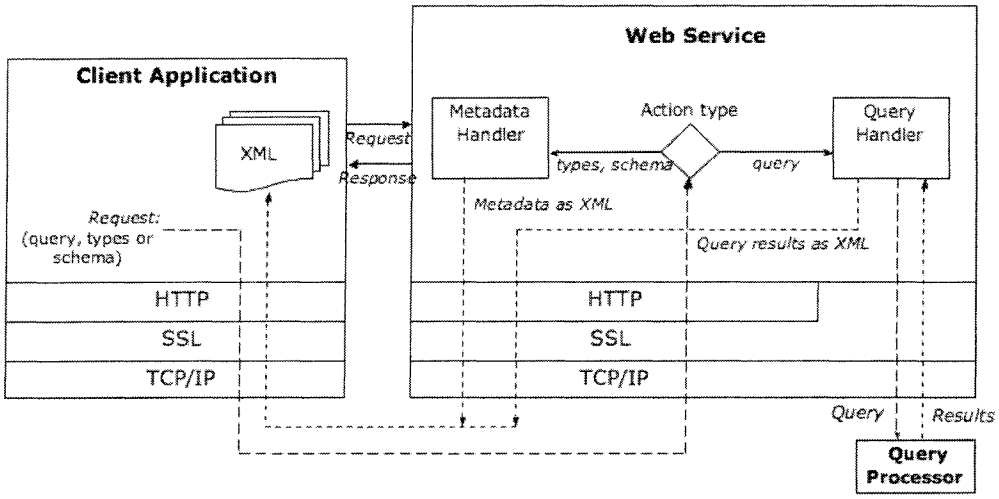

Figure 3.1. System architecture: client and web service

When dealing with large static datasets, b+ trees provide a good tradeoff between storage overhead and search performance as shown in section 3.6. DiSARM uses Berkeley DB databases to index data streams in real-time. Data stream indexing can be split across multiple nodes for scalability. Periodically, Berkeley DB instances are aggregated from multiple nodes and converted into static b+ trees. The query processor can access both the Berkeley DB databases and static b+ trees to provide real-time access to multi-terabyte datasets.

The web service satisfies requests for schemas and data types via the metadata handler. Queries submitted to the servlet are validated and submitted to the query processor in the *Query Handler* module. The query processor returns results to the servlet via the *Result Processor Module*, which converts raw data records into an XML document to return to the client. By acting as a proxy, the web service module was added to the existing system without having to modify the query processor. Because data in the system is stored in its original state, data is converted into an XML document each time a result document is generated. Section 3.6 discusses the performance penalty for this conversion and shows that the tradeoff between storage overhead and query performance is acceptable.

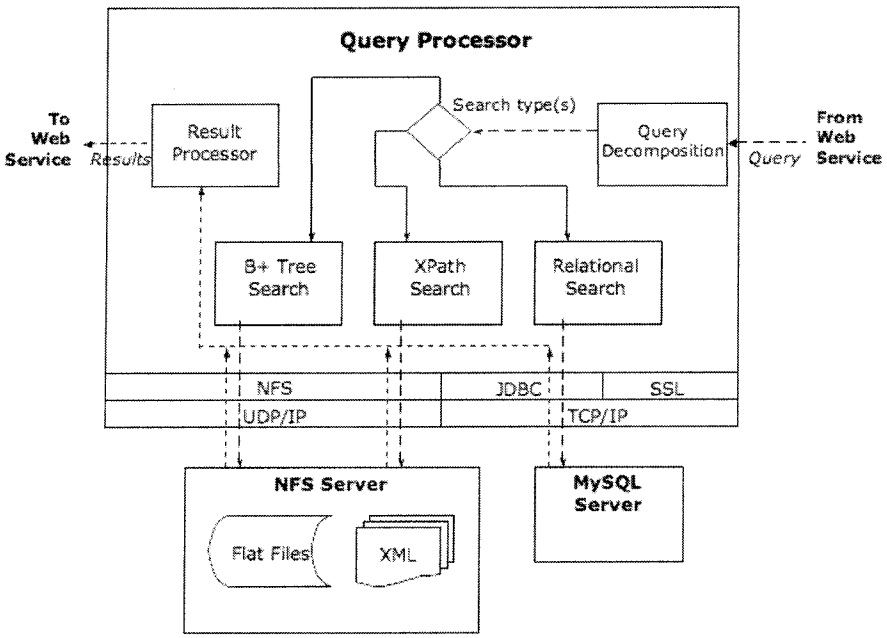

Figure 3.2. Query processor architecture

The DiSARM web service is implemented as a Java servlet that responds to the client on behalf of the query processor, which is detailed in figure 3.2. The base web service specification has been implemented in DiSARM according to sections 3.3 and 3.4. Each HTTP request to the web service provides an *action* parameter with one of three values: "query", "schema" and "types". These parameters are summarized in table 3.5. The *schema* action returns the query DTD or the result DTD as specified by the additional *type* parameter. The *types* action returns an XML document that defines the data sources that are made available for querying through the web service. Table 3.6 describes three additional parameters for the *query* action: "query", "limit" and "total". Only the *query* parameter is required by the API definition when specifying the query action. The value of the *query* parameter is the XML query to be executed by the web service. The remaining parameters, *limit* and *total*, were implemented at LANL to provide additional functionality, optionally limiting the number of records returned to the client by the web service.

Requests are processed by the web service as follows: the client opens an HTTP connection to the service and submits a POST request containing the action to perform along with any additional parameters. If the schema or types action is requested, the web service can complete the request. If the query action is requested, the servlet passes the query to the

query processor, which can access flat files, XML documents or relational databases depending on how the requested data is stored.

Table 3.5. Action parameters

Parameter Value	Description	Additional Parameters
query	Return results, as XML per the result DTD, for the XML query contained in the *query* parameter	See table 3.6
schema	Return either the query DTD or the result DTD as per the *type* parameter	type=query or type=result
types	Return the data types and sensors, as XML per the result DTD, that can be queried at this site	None

Table 3.6. Parameters for query action

Parameter Value	Description
query	The query to be executed as an XML document
limit	The maximum number of records to return per data type
total	The maximum total number of records to be returned; *total* >= *limit*

The DiSARM query processor is able to perform parallel query execution by decomposing queries by data type, sensor and time range. After decomposition, the resulting queries can be sent to query processors on remote nodes. The query processor that received the request from the web service layer then aggregates results from the nodes and sends the results to the web service.

Large flat file data sources are currently indexed using b+ trees and stored on NFS mounted RAID arrays. These structures have been designed to offer a performance trade off between low overhead and query power. The key values currently supported are source IP address, destination IP address, destination port and start timestamp. Each key value maps to a file offset where a record matching the key value is stored. MySQL is used for smaller data sources, including IDS, intrusion prevention system (IPS) and anomaly detection system

(ADS) alerts. By using DiSARM, analysts can query these different data storage technologies in a uniform manner.

3.5.1. Security features

Security in DiSARM is central to the system's design and implementation, as shown in figure 3.3. The system has integrated security elements from the bottom up, providing an effective solution for storing sensitive data sources. Table 3.7 summarizes the security features native to the prototype system.

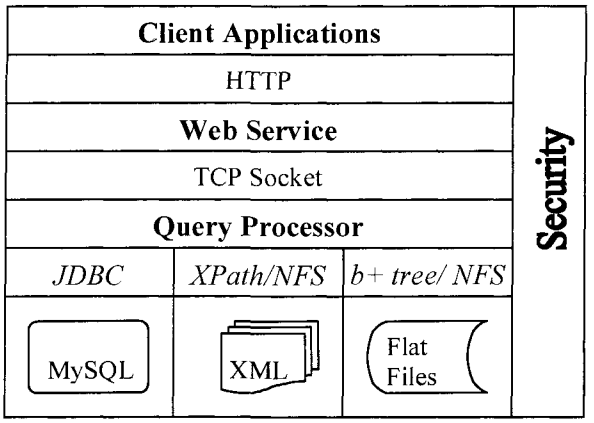

Figure 3.3. Generalized system architecture

Table 3.7. Security features

Component	Security Feature(s)
Client Applications	Communicate with web service via SSL; provide user name and password
Web Service	Communicate with client and query processor via SSL
Query Processor	Communicate with web service via SSL; authenticate user; enforce per-record access controls
JDBC	Connection to RDBMS requires username and password; Only a fixed set of hosts can connect to server
XPath	None
b+ tree	File system permissions and NFS exports restrict access to query processor
MySQL	Permissions set to enforce per-host user authentication to query processor

The client must connect to the web service via SSL to initiate a request. The web service must then submit the request to the query processor. In each case, communications are via SSL to protect both the query content as well as the results. Clients must provide a valid SSL certificate to open a HTTP connection to the web service. This also holds true for communications between the web service and the query processor, which can reside on different servers. In this case, raw TCP sockets are protected with SSL instead of an HTTP connection.

The query processor implementation provides authentication and access control via the *user* and *password* attributes in the root *Query* element of each query to enable authentication at the query server. The system is integrated with a one-time password authentication service for interactions with users and a traditional reusable password mechanism for interacting with applications utilizing the web service.

Once the query processor authenticates the client, the user credentials are used to enforce access controls to data in the system on a per-record basis. For example, a system administrator might be responsible for a set of servers in a particular network subnet. The access controls for this user can be configured to only allow access to records relevant to that subnet. The query engine dynamically filters out any query results not pertaining to the subnet prior to sending data to the web service. As with SSL, the trade off between privacy and performance is a factor. Checking each record in a result stream can be an expensive operation depending on the number of access controls being enforced. This feature is optional, as users can be configured to have no restrictions, thus eliminating the need to check individual records.

Communications with MySQL servers occur over JDBC and are restricted to only communicate with the query processors through MySQL's security settings. Although XPath does not provide any security measures, the underlying XML documents are protected by file permission restrictions and NFS export control. The b+ tree data structures and flat file logs are protected in the same manner.

Lastly, flat file logs and XML documents are protected by automated file integrity checking to ensure the underlying data is not manipulated or corrupted. A SHA-1 checksum of each file is periodically generated and compared against an existing checksum stored in a

secure MySQL database. An alert is sent to the system administrator if the checksums do not match, along with the time at which the file last passed its integrity check.

3.5.2. Testing

To test the prototype system, unit testing was performed from the bottom up on each system component. The b+ tree implementation was written in 2002 and has been tested rigorously by LANL's network security team for a number of years. The query processor was built on top of the b+ tree implementation to access multi-terabyte datasets. Access to data sources in MySQL databases and XML documents has been added to the query processor as needed.

Each time a new data source is integrated into the system it is implemented and tested independent of the existing system. This allows for efficient unit and case testing of the new component without the need of testing the entire system. Typically, a web interface is built on top of the query processor for the new data type and tested by analysts who report any bugs or data inconsistencies to the lead developer.

After a successful testing period with the web interface, a new data type can be integrated into the web service. At this point, as much as possible of the web interface code is reused and integrated into the web service. To test the XML conversion code, an XML parser validates several test case result documents. Then, these same documents are compared with results generated via the web interface.

3.6. System Evaluation

This section presents the performance of our prototype system. First, section 3.6.1 details scalability of the prototype implementation, by showing its strength over MySQL for high-throughput data capture. Next, section 3.6.2 shows that our b+ tree implementation requires much less storage overhead than MySQL, as our approach keeps only a single online copy of the data. Section 3.6.3 show the query performance of our implementation compared to MySQL for a limited set of queries. Lastly, section 3.6.4 presents the overhead imposed on the system by implementing the web service layer on top of the existing query processor.

3.6.1. Data capture performance

Data capture is an important of the DiSARM architecture. Any system attempting to capture and provide access to large network traffic datasets must be able to scale to meet the needs of a dynamic and rapidly growing network. DiSARM uses Berkeley DB to index data streams across multiple nodes in real-time, allowing the system to scale. This section shows the performance benefits of this approach as compared to using a relational database, in this case MySQL.

NetHead data from two days was used to evaluate the throughput of the system prototype. The first dataset was from April 2002, containing approximately seven million records. The second dataset was from September 2003 and contained approximately 20 million records. The large difference in the number of records can be attributed to an increase in network activity coinciding with the Microsoft DCOM vulnerability. These datasets were chosen to illustrate how rapidly the size of the datasets grows. Due to the increase in port 135 and 445 scans, the number of NetHead records tripled over a one-week period. Until the number of DCOM scans increased so dramatically in late August 2003, the April 1 dataset was very representative of a typical day's worth of NetHead data.

The primary design decision that impacts throughput is the backend storage mechanism. MySQL (version 3.23.53) was chosen for comparison because it is widely used in intrusion detection systems like Snort and thus many intrusion detection analysts have experience in its operation. Two dual Pentium III 1 GHz computers with 1 GB RAM and gigabit Ethernet were used for the test. For the MySQL evaluation, records were inserted one at a time into an empty table and the default table and database options were used. A client program on one node sent each record to a MySQL server on the second node. For the DiSARM evaluation, one node streamed data to a DiSARM server on the second node, which indexed the data stream. Figure 3.4 illustrates the results of this benchmark.

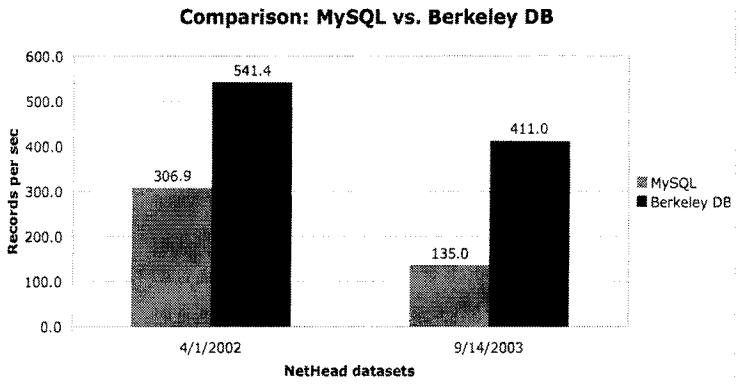

Figure 3.4. Backend data store performance

Although the throughput rate was acceptable for the 2002 data, the throughput fell 46% on the 2003 data to an average of 135 records per second. At this rate, it took 1.6 days to load the dataset into MySQL. By comparison, the same test using Berkeley DB (version 4.1.24) yielded superior throughput while only showing 24% degradation in performance on the larger dataset. By contrast this dataset took only 13 hours to load using Berkeley DB. Using MySQL's bulk data load capabilities yield improved results although these results are not applicable as the real-time requirement of the system is not satisfied.

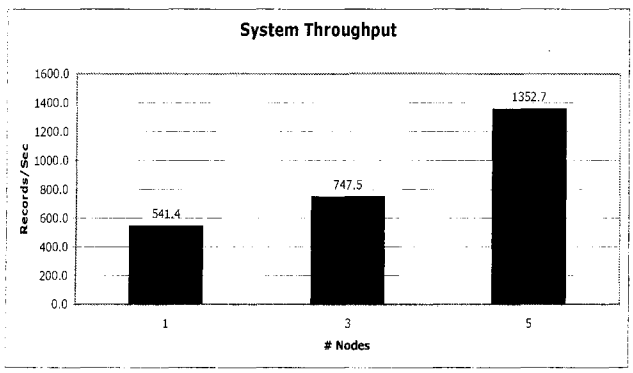

Figure 3.5. System throughput

Figure 3.5 illustrates scalability, another important aspect of system performance. To deal with rapidly growing datasets, the system must be able to scale. For this benchmark, three DiSARM configurations were evaluated: single-node, three-node and five-node. In the three and five node setups, data was sent to one node and then split to the remaining nodes

which indexed the data stream in parallel. As shown, the system scaled well over the 2002 dataset as additional nodes were added for indexing.

3.6.2. Storage performance

The desire to keep storage overhead at a minimum was a key driver in the design and implementation of DiSARM. Two long-term storage methods were examined: relational databases, in this case MySQL, and static b+ trees. Both approaches have advantages and disadvantages. Relational databases provide the user with a great deal of power in manipulating the data. The drawback is that the overhead is high and it introduces data redundancy. One of the design goals was to keep a single online copy of large datasets. Storing each record in a relational database and storing the raw, or *native*, data dramatically increases the amount of online data. To avoid this, static b+ tree indexing was implemented in DiSARM. The advantage of this approach is that the native data is indexed, eliminating the need for an additional online copy of the data. The disadvantage is that the analyst no longer can leverage the power of a relational database.

For this benchmark, three cases were examined: a relational database with native data, a relational database without native data and static b+ trees indexing native data. MySQL 3.23.53 was used on Mac OS X version 10.2.7. For the b+ tree and MySQL databases, four fields were indexed: destination address, source address, destination port and start timestamp. As is evident in table 3.8, static b+ trees require 45.5% less storage than using MySQL as the access method without storing the native data. Keeping an online copy of the native dataset and storing it in MySQL as well requires 277.5% of the storage compared to only keeping the native dataset online.

Table 3.8. Storage overhead

Storage Method	Size (MB)	% Diff
Native	858.4	100.0
Native, Indexed	1124.5	131.0
MySQL	1523.6	177.5
MySQL, Native copy	2382.0	277.5

3.6.3. Query performance

To compare the b+ tree query performance to MySQL, five different queries were run five consecutive times on a dual Xeon server with 2 GB of RAM. MySQL version 4.0.18 was used for comparison in this test. Table 3.9 illustrates the averaged results from each query.

Table 3.9. Query performance

Query	b+ Tree (time in seconds)	MySQL (time in seconds)	% Diff	# Results
1	7.84	0.45	5.69%	18
2	0.37	0.03	6.94%	184
3	0.36	1.60	448.37%	758
4	98.16	1,012.94	1,031.93%	820,410
5	136.53	1,831.13	1,341.19%	369,186

For brevity and confidentiality, the query syntax used to generate the above results is not included. Query 1 returned results matching on a source address and destination port. Query 2 returned results from the same destination port as query 1 but with no restriction on source address. Query 3 returned results matching on a small range of destination ports. Query 4 returned results matching a highly active destination port. Query 5 returned results matching a larger port range than Query 3.

As is evident in table 3.9, the b+ tree implementation performed well for cases generating a large number of results. The results from queries 1 and 2 were substantially better when using MySQL, as the caching mechanism helped consecutive runs perform very well. Even so, the total time to return these queries was low when using either system.

These initial results support the decision to use the b+ tree data structure as the underlying data retrieval mechanism. Performance is superior to MySQL when dealing with large volumes of data, as was the initial design goal of the system. It is also reassuring that queries with small result sets also return quickly, although not as fast as when using MySQL in some cases.

3.6.4. Web service performance

The evaluation of the DiSARM web service layer focuses on the additional overhead incurred by the web services layer. As discussed in the implementation section, the web service servlet receives queries from the client over HTTP and sends the query to the query processor. Results are returned to the servlet in their native format or as comma separated value records in the case of data types stored in relational databases. The servlet parses each record according to its type and outputs an XML representation of the record back to the client over HTTP.

Two factors are measured in the evaluation: query performance and storage overhead. Six different data sources are used for the evaluation: TippingPoint UnityOne Intrusion Prevention Appliance block logs, R3000 web logs, NetHead connection records, Snort IDS alerts, Estimated Moving Average Anomaly Detector (EMAAD) anomaly alerts and LFAP flow records. NetHead is a connection-based intrusion detection and network forensics tool developed at LANL. EMAAD is an unsupervised statistical anomaly detector also developed at LANL that is adept at discovering hosts exhibiting unusual scanning activity. The queries used in the evaluation have subnet information obfuscated using asterisks. For both tests, two datasets were collected: one where the client connected to the web service via HTTP over SSL and another where the client connected to the query processor via a raw TCP socket over SSL. By comparing these results we show the overhead incurred by the web service implementation.

Each evaluation query shown in table 3.10 was executed five times, both with a client connecting to the query processor and with a client connecting to the web service. The same test was recorded with the client reading between 1000 and 10,000 results, varying by 1000 results for each test. This test shows that there is a fixed cost associated with processing results through the web service layer. The query timer was started when the first record was available to be read by the client. The timer was stopped when the last record, again between 1000 and 10,000, was read. The choice to time queries after the query processor had computed the result set was made as the query processor is processing the same query if the client connects via the raw socket or the web service and thus does not need to be included in the overall time of the test.

Table 3.10. Evaluation queries

Type	Query	Results
TippingPoint	<Query srcip="128.165.0.0~128.165.255.255"> <Timestamp start="2005-7-11" end="2005-7-17"/> </Query>	55,624
R3000	<Query srcip="128.165.***.0~128.165.***.255"> <Timestamp start="2005-7-18"/> </Query>	48,407
NetHead	<Query srcip="128.165.***.0~128.165.***.255"> <Timestamp start="2005-7-18"/> </Query>	39,578
Snort	<Query srcip="128.165.0.0~128.165.255.255"> <Timestamp start="2005-1-1" end="2005-7-31"/> </Query>	10,321
EMAAD	<Query srcip="128.165.0.0~128.165.255.255"> <Timestamp start="2004-1-1" end="2005-7-31"/> </Query>	12,725
LFAP	<Query srcip="128.165.***.0~128.165.***.255"> <Timestamp start="2005-7-18"/> </Query>	393,993

The goal of this test is to show the relative performance of clients connecting to the web service versus connecting directly to a socket on the query processor. As shown in figure 3.6, the TippingPoint query results show the smallest performance penalty at 2.6%. The remaining datasets showed reasonable performance although the LFAP dataset was 12.1% slower compared to the raw socket connection.

The second area of evaluation is the performance penalty incurred by converting data from its native format into the web service's XML representation. Two factors are tested: processing time overhead and storage overhead. Figure 3.7 shows the processing overhead and figure 3.8 shows the storage overhead for the example query.

The processing overhead results show the additional percentage of processing time for converting it into the XML document. The application used in the evaluation reads the results from the evaluation queries into memory and then times how long it took to convert the records into the XML document. The application used the same XML conversion algorithms as the web service and sent the results to /dev/null. As shown, the processing overhead

associated with XML conversion is negligible on a per-record basis. The more important factors are I/O and parsing records into objects.

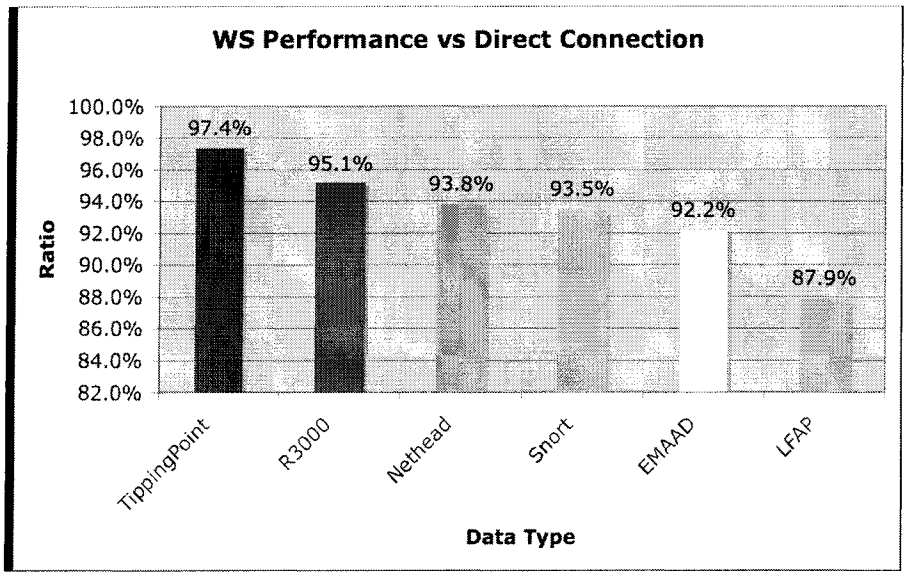

Figure 3.6. Web service performance overhead

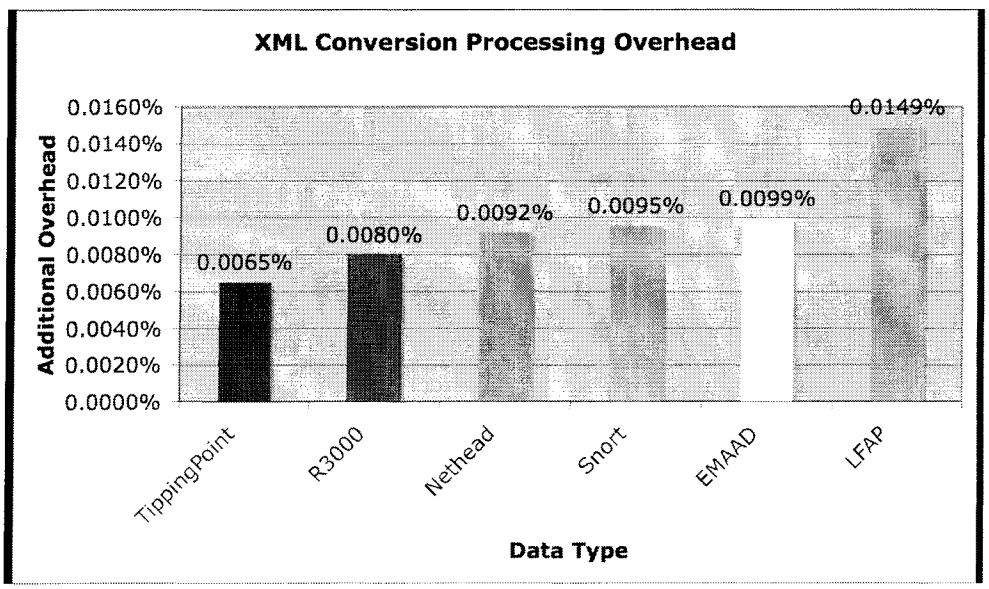

Figure 3.7. Processing overhead

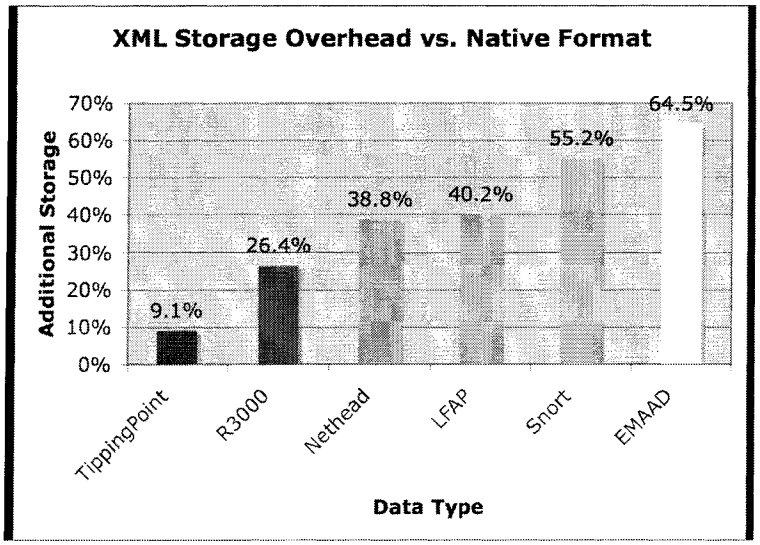

Figure 3.8. Web service result set overhead

Storage overhead is important to evaluate, as XML data is larger than data in its native format due to the addition of tags. This can impact both memory usage and disk usage, both of which are important concerns when dealing with large volume data sources. Our next evaluation shows the amount of additional storage needed to store the query results of the evaluation queries in the XML format compared to storing the results in their native format. These results are encouraging, as the data sources with the largest relative overhead, Snort and EMAAD, were the lowest volume data sources in our evaluation environment.

3.7. Summary

In summary, we present a design and implementation that provides analysts with the tools necessary to perform network security investigations over multi-terabyte datasets across multiple sites using web services. A domain specific query language using XML documents to represent queries of network security data source is defined along with a generalized XML document format for returning query results. We also define the API needed to exchange data between remote sites and understand the supported result formats.

In evaluating the prototype system we have provided results for data capture throughput, query performance, storage overhead and overhead imposed by the web service

layer. Scalable, high-throughput data capture and retrieval has been shown using the NetHead dataset as a test case. Furthermore, the static b+ tree indexing implementation minimizes storage overhead yet still provides analysts with a powerful tool for querying data from heterogeneous datasets. Finally we show that the overhead of utilizing web services is acceptable for the prototype system.

4. A Framework for Alert Verification and Event Correlation

As attacks on networks continue to grow in volume and complexity, IDSs research has begun to focus on alert correlation to address the inherent limitations of traditional IDSs. In this chapter we present a framework design and implementation that provides a scalable solution for two important components of alert correlation: alert verification and event correlation.

In our framework, a broker application maintains an alert database containing IDS alerts while distributed software agents perform alert verification and event correlation of alert instances. Agents can be autonomous or event-driven and are designed to run on multiple hosts to ensure scalability of complex tasks.

In our implementation, agents communicate with the broker via a web service architecture, making them easy to build and deploy in heterogeneous networks. In our implementation, three IDSs from multiple paradigms are supported, showing that the framework can be applied to various IDS types.

4.1. Introduction

Alert correlation has become an active research area in network security and intrusion detection due to the growing volume and complexity of attacks on networks. Correlation is used to reduce the number of low-level, or elementary, alerts and provide knowledge discovery of complex patterns and relationships in alert datasets.

Research in the area of alert correlation is closely tied to the work described in this chapter. Correlation techniques invariably use some degree of expert knowledge to establish how alerts can be linked. Valdes and Skinner use a probability matrix to determine how incoming alerts should be correlated within the EMERALD system [15]. This technique showed to be effective on the 2000 DARPA dataset although the accuracy of the probability matrix was essential for adequate performance. Another system used the JIGSAW predicate language to represent alerts as *hyper-alert correlation graphs* [16] and requires domain knowledge and network facts, such as host and vulnerability information, to provide effective

correlation. Similar work by Cuppens and Miège uses the LAMBDA specification language to achieve correlation [10].

As with any system, the quality of data, in this case intrusion detection system (IDS) alerts, determines the quality of the results. The outcome of alert correlation depends on IDS alerts with low false positive rates. Frequently occurring false positives in the dataset lead to inaccurate results regardless of the algorithm used for alert correlation.

To reduce false positives, an *alert verification* phase is necessary prior to correlation. This process is highly dynamic and dependent on domain knowledge of network security. An example of alert verification is correlating vulnerability information with a known IDS signature. If the target host is vulnerable to the detected attack, the alert can be ranked ahead of other similar alerts. Likewise, if the host is not vulnerable to the attack, for example an exploit for a Windows vulnerability on a Linux system, the alert can be marked as a false positive.

Event correlation is another important aspect of the alert verification process. The previous example uses correlated vulnerability information to verify the alert. This is an example of event correlation, the process of finding information related to an IDS alert that is relevant in its analysis. Event information can be found in logs or other information sources such as a network knowledge base.

Additionally, when discussing alert verification and event correlation, scalability and flexibility must be considered. If a system cannot scale to meet the demand of large production networks then the techniques, no matter how useful, will not be used in practice. Likewise rigidly designed or monolithic systems cannot adapt to highly dynamic networks. A flexible, yet scalable, solution is needed to address these limitations. Valeur et al. describe a framework that incorporates alert verification and event correlation functionality in [35]. Our framework is designed to contribute to existing research in this area by improving upon limitations of the existing work, such as scalability and flexibility.

We describe a framework that addresses four key areas: alert verification, event correlation, scalability and flexibility. Software agents are deployed within the framework to provide alert verification and event correlation capabilities. Agents are autonomous and can reside on multiple hosts ensuring scalability. Alert information is stored and maintained by

an alert broker. Agents and the alert broker communicate via a web service interface, allowing the framework to span network boundaries and making the system flexible and easy to deploy.

Our framework is lightweight yet has a powerful set of features. The *broker* receives new IDS alerts from *alert agents* that gather alerts from the IDS sensors. The design of our system is IDS independent, supporting alerts from misuse IDSs, ADSs and intrusion prevention systems (IPSs).

The broker also handles requests and responses from *evaluation agents*. Evaluation agents are asynchronous processes that request alerts from the broker as needed for alert verification and event correlation tasks. The broker determines which alerts to send to the agent based on event driven rules, allowing for hierarchical processing of IDS alerts. Computationally expensive evaluation agents can depend upon actions of lightweight agents to assure that the system does not perform unnecessary computation. For example, an evaluation agent can be written to check that a network connection was successfully established between the target host and the attacker during the attack indicated in the IDS alert. The broker assures that agents dependent cannot process the alert unless the agent finds evidence that a connection was established.

Agents can send three types of information to the broker: alert assertions, correlated events and informational messages. Alert assertions are made by the agent to apply a label, such as *false positive* or *valid*, to an alert. The broker maintains an entry per alert for each agent's assertion, which can be abstracted to represent an overall ranking for the alert. Correlated events can be gathered from any number of data sources by the agent and returned to the broker. The broker stores this information, which can be used for a variety of tasks in analyzing and processing of IDS alerts. Informational messages are attached to alerts in the broker and provide a means for agents to add free text comments to individual alerts. For instance, an agent asserting an alert as a false positive can add informational messages explaining to the analyst how the agent came to this conclusion.

Our prototype framework implementation uses web services for communication between the broker and agents. Alerts from alert agents and correlated events returned by evaluation agents are stored in a relational database and can be exported as XML. At LANL,

we have deployed several evaluation agents that provide alert verification and event correlation capabilities for three IDSs used in practice.

4.2. Framework Design

The alert evaluation and event correlation framework, as shown in figure 4.1, is comprised of four entities as shown in: *alert agents*, an *alert broker*, an alert database and *evaluation agents*. Alert agents send alerts from IDSs to the alert broker. The alert broker stores alerts in the alert database and sends alerts to the evaluation agents when requested. Evaluation agents are autonomous software agents that request alerts from the broker, process these alerts and return pertinent information to the alert broker.

Evaluation agents perform functions of event correlation and alert verification. An evaluation agent is further classified as an *event correlation agent*, an *alert verification agent* or a *hybrid* agent depending on the functions performed by the agent. Event correlation agents process IDS alerts by locating events in log records or other information sources that are related to the IDS alert and send the correlated events to the broker to store in the alert database. Alert verification agents use expert rules and corroborative information to assert the validity IDS alerts. Hybrid agents perform both event correlation and alert verification, as illustrated by the overlapping dashed rectangles in figure 4.1.

Communication between the broker and agents is handled by messaging, allowing for entities in the framework to reside on multiple hosts for scalability and flexibility in deployment. The technical details of message passing are dependent on implementation. The prototype system described in section 4.3 uses a combination of HTTP-based web services and XML information content for messaging and information exchange.

The following subsections are arranged as follows: section 4.2.1 describes the alert agent design, section 4.2.2 describes the alert evaluation agent design and section 4.2.3 describes the alert broker design.

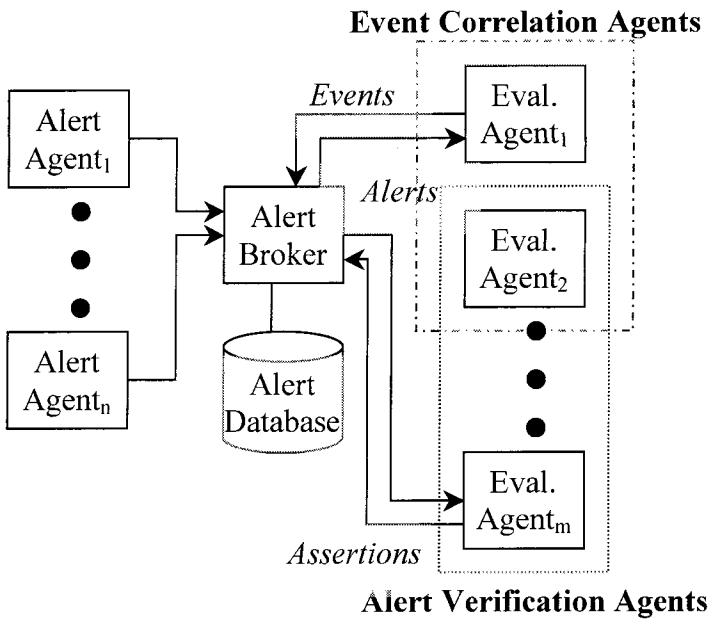

Figure 4.1. Framework architecture showing an alert broker with *n* alert agents and *m* evaluation agents

4.2.1. Alert agent design

Alert agents are the processes responsible for sending IDS alerts to the alert broker. The agent has two functions: obtaining state information from the alert broker and sending alerts.

When an alert agent is initialized, it obtains state information from the alert broker to determine which alerts to send. The agent maintains an ordered list of alerts with unique alert identifiers defined by the agent. Alerts are placed on the list as they are generated by the IDS and are removed from the list when sent to the broker.

The list is needed to maintain synchronization with the alert database by assuring that all alerts generated by the IDS are sent to the broker exactly once. To obtain state information, the agent sends a *state* message to the broker. The broker returns the last alert identifier sent by the agent.

After obtaining the state information from the broker, the agent can send new alerts to the broker. The next alert to send is the alert in the ordered list that follows the alert identifier returned by the broker. This may or may not be the first element of the list depending on

whether the agent exited safely prior to being restarted. If the list is empty, this implies that all alerts from the IDS have been sent to the broker and the agent must wait until new alerts are added to the list.

To send an alert, the alert agent sends an *alert* message to the broker that includes the alert identifier and information content. The broker stores this information in the alert database and replies with a success indication if the operation was processed correctly. An error message is returned if the message could not be processed. If the operation was successful, the alert agent removes the alert from the ordered list and continues by sending the next alert.

4.2.2. Alert evaluation agent design

To process an alert, an alert evaluation agent requests an alert from the broker by sending a *next* message. The broker determines which alert the agent needs to process next, a process that is described in section 4.2.3. The broker sends back the information content of the next alert, along with a unique identifier, that needs to be processed by the agent. The agent then processes the alert and sends back one or more responses to the broker.

An evaluation agent provides the alert broker with three types of information: *general* information, *assertions* and correlated *events*. The three types are shown in table 4.1 and are sent to the broker as an *information* message with a parameter identifying the type of information that is being added to the alert.

Table 4.1. Types of information returned to the broker by alert evaluation agents.

Info Type	Description	Data Type
general	Miscellaneous information about alert processing or free text comments for users	Text
assertion	A floating-point number between -1.0 and 1.0; positive values indicate a *valid* alert; negative values indicate an *invalid* alert; zero indicates *unknown*	Scalar
event	An event record or piece of information related to the alert, as determined by the agent	Record content

General information is sent to the broker to add contextual, free-text information about the processing that is being performed on the alert. The agent sends a message to the broker that includes the alert identifier and the information content that is being added. For example, the agent could return information on the execution time of a particular alert evaluation task, which could be used by a security analyst to tune the agent.

Assertion information is returned by an evaluation agent to provide the alert broker with information on the validity of the alert. The value returned by the agent is a floating-point number between -1.0 and 1.0, with a positive value indicating valid alert and negative values indicating an invalid alert. The value zero is returned if the agent cannot verify the alert for any reason. The agent's determination of what constitutes a valid or invalid alert is dependent on the particular agent implementation.

By using floating-point values for verification, agents can be implemented to return validity assertions with a confidence factor. For instance, an agent can return a value of 0.8 if it is 80% certain that an alert is valid. This is useful in practice as in many cases the verification process is influenced by low quality or missing data.

Event information is returned by an alert correlation agent to provide the broker with information related to the alert that may be of use to processes and users analyzing the alert. Correlated events are stored in the alert database by the alert broker. As with assertion information, gathering event information is dependent on the agent's implementation. In sections 4.3.4, 4.3.5 and 4.3.6 we present examples of event correlation agents that have been implemented in our prototype system.

Evaluation agents send a *complete* message to the alert broker when processing of an alert is complete. By using a separate message to signal completion, the agent can send multiple information messages to the broker if needed. The ability to send multiple messages allows for agents to perform tasks of alert verification, event correlation or a combination of the two, providing flexibility in the design of alert evaluation agents.

Another benefit of the *complete* message is to allow agents to operate asynchronously. When an alert is sent to an evaluation agent, the broker places the alert in the *in progress* state. The broker takes no action on this alert until the evaluation replies with a complete message at a later point in time. While in the *in progress* state, the evaluation agent can take

any necessary action and requires no additional interaction with the broker for the alert until the *complete* message is sent whereupon the broker sets the alert state to *processed* indicating that the alert has been processed by the agent. Alert state management is described in the following section.

4.2.3. Alert broker design

The alert broker is designed to handle messages from alert agents and alert evaluation agents and maintain the alert database. The broker also maintains a list of alert and evaluation agents and must be capable of uniquely identifying and authenticating each entity. This is necessary as the broker must be able to associate messages with agents and also assure that only valid agents are interacting with the broker. Managing and identification of agents is implementation dependent.

The messages in table 4.2 show the broker's functionality and have been described in sections 4.2.1 and 4.2.2 above. The remainder of this section provides additional information specific to the alert broker for the processing of these messages.

An alert agent sends the *state* message to the broker upon initialization. The broker maintains an entry storing the last agent-defined alert identifier for each alert agent associated with the broker. The identifier is returned to the agent so that it can restart processing at the point of the last successfully processed alert. The identifier is updated upon successful completion of an *alert* message.

The *alert* message is sent by the alert agent to the broker containing alert content from an IDS associated with the alert agent along with the agent-defined alert identifier. The broker is responsible for parsing and storing the alert content in the alert database. If the alert is successfully processed, the broker updates the agent's alert identifier entry and returns a success message to the alert agent. If the alert cannot be processed, an error message is returned to the agent.

The *next* message is sent to the broker by an alert evaluation agent to request the next alert to be processed. The alert broker maintains an ordered list of alerts to be processed by each agent. The first alert is sent to the agent in the *next* reply and removed from the list.

Table 4.2. Summary of alert broker messages

Message	Description	Parameters	Return Values
state	Sent by alert agent requesting alert processing state	None	Alert identifier
alert	Sent by alert agent sending an IDS alert to the alert broker	Alert identifier; alert content	None
next	Sent by evaluation agent to request a new alert for processing	None	Alert identifier; alert content
information, general	Sent by evaluation agent to provide miscellaneous free-text information about an alert	Alert identifier; information content	None
information, assertion	Sent by alert verification agent to assert that an alert is valid, invalid or unknown	Alert identifier; assertion value	None
information, events	Sent by event correlation agent to associate one or more events with an alert	Alert identifier; event(s) content	None
complete	Sent by alert evaluation agent when processing of an alert is complete and all information messages have been sent	Alert identifier	None

The maintenance of the alert list is implementation specific but allows for the creation of *agent dependencies* in the broker. Each alert in the alert database has a state associated with each evaluation agent that is maintained by the alert broker. Alerts can be labeled "in progress", "processed", "conditional" or "available". An "in progress" alert is an alert that has been sent to the corresponding evaluation agent via a *next* message and a "processed" alert is an alert that has been evaluated by the agent and followed by a *complete* message. A "conditional" alert is an alert that has not been sent to the evaluation agent, as it is dependent on another agent's evaluation of the alert. Alerts labeled "available" are those ready for processing by the evaluation agent and are the only types of alerts that can be placed on the agent's list of alerts. Table 4.3 summarizes the alert states.

The alert state diagram is presented in figure 4.2. State S_1 transitions to S_2, the *conditional* state, when an evaluation agent has dependencies on one or more evaluation agents. If the agent is independent the state transitions to S_3, the *available* state. The state transitions to state S_4 after the alert is sent to the evaluation agent in a *next* message. The alert transitions to state S_5 following a *complete* message from the evaluation agent.

Table 4.3. Alert state descriptions

Alert State	Description
in progress	The alert has been sent in a *next* response to an evaluation agent for processing
processed	The evaluation agent has signaled that the alert has been processed by sending a *complete* message
conditional	The alert cannot be sent to the evaluation agent as an agent dependency exists for the alert
available	The alert has no outstanding dependencies and is available for processing by the agent sending a *next* message

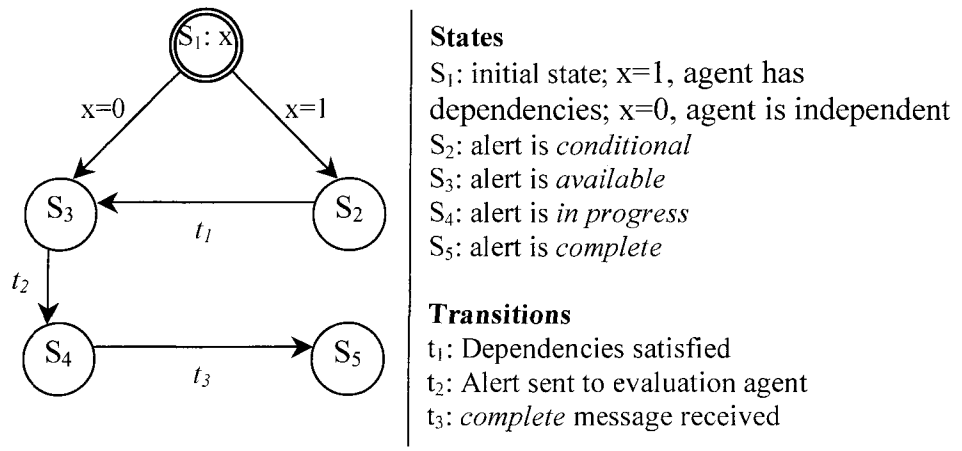

Figure 4.2. Alert state diagram

As individual alerts have a state associated with each evaluation agent, an alert can exist in multiple states. The broker uses this information to build agent dependencies. For example, an evaluation agent *A* that depends on evaluation agent *B* will have its alert states set to "conditional" when they are initialized in the alert database. As *B* sends *complete* messages for alerts it has processed, the corresponding entries in *A* are changed from "conditional" to "available", indicating that they can now be processed by *A*. Determining the conditions of agent dependencies is a key function of the broker and is dependent on its implementation.

As discussed in section 4.2.2, there are three types of *information* messages, which are shown in table 4.2. The *general* information message contains free-text information or data

to be associated with the alert identified in the message. The broker stores this information in the alert database.

The *assertion* information message is sent by an alert verification agent to indicate the validity of an alert. The scalar value returned to the broker is stored in an entry in the alert database for the agent's evaluation of the alert identified in the message. The alert state is updated to "processed" to indicate that the alert agent's evaluation is complete.

The *event* information message is sent by an event correlation agent to associate events with the alert identified in the message. One or more events can be sent in a single *event* message depending on implementation.

Lastly, the *complete* message is sent by an evaluation agent after it has finished processing the alert. The alert broker updates the state of the alert identified in the message from "in progress" to "processed". This information can be used in defining agent dependencies to allow dependent agents to process the alert once it is completed.

4.3. Implementation

This section describes the implementation of the alert verification and event correlation framework developed at LANL. Section 4.3.1 provides a general overview of the system. Section 4.3.2 discusses alert state. Section 4.3.3 presents the alert dependency implementation. Sections 4.3.4, 4.3.5 and 4.3.6 describe example evaluation agents developed for specific operational tasks at LANL. Appendix C provides a summary of relations used in the implementation.

4.3.1. Overview

The alert verification and event correlation framework has been implemented as a prototype system for LANL's network security team. The alert agents, alert broker and evaluation agents are written in Java, although they can be written in any language as the use of web services makes implementation language independent. The alert broker handles the web service messaging and is a servlet deployed using Apache Tomcat. The broker also maintains the alert database, which is a MySQL database on the broker's server.

Agents communicate with the web service by sending HTTP requests to the server specifying an agent identifier, message type and message content as parameters. The system is deployed on a closed, protected network, alleviating the need for authentication in the prototype although future implementations will include this capability. Sending messages over HTTPS or digitally signing messages are possible solutions to provide authentication in environments where trust cannot be assumed in the deployment.

The broker's reply is returned in the HTTP response as text. Each line of the response is of the form "*name*, *value*" where the fields represent the parameter's name and value. This response format is adequate, as the responses sent by the broker are straightforward. Future versions of our implementation will utilize web service standards like SOAP to allow for more powerful messaging capabilities.

Alert agents have been developed for TippingPoint IPS alerts, Snort IDS alerts and Estimated Moving Average Anomaly Detector (EMAAD) ADS alerts. Snort and TippingPoint are used in many network security environments. EMAAD is a proprietary anomaly detector developed at LANL to monitor hosts for anomalies using network flow records.

While a single alert broker instance can manage alerts from the three IDSs simultaneously, our prototype deployment uses multiple broker instances to manage Snort alert and EMAAD alerts separately. A third instance has been deployed to store all three alert types using the same broker. The reason for creating multiple broker instances is to allow various sets of evaluation agents to be designed, implemented and tested using the different configurations. The three instances use the same code base and differ only in the alert types stored in the alert database and the types of evaluation agents managed by each broker.

Information content returned to the broker in a general information message is stored in a *general information* relation as it was sent in the HTTP request. Correlated events are returned to the broker as XML and stored in a *correlated events* relation. Replacement of all standard HTML characters with their meta-character equivalents is necessary prior to insertion in the database (e.g. "<" is replaced with "%lt;").

Alert information is stored in four relations: a *universal alert* relation and a *type-specific alert* relation for each of the three supported IDSs. The universal alert relation stores

miscellaneous information about the alert used by the broker in processing and the type-specific relation stores the alert content. A unique alert identifier is the primary key across the four relations and is used when performing joins between the universal alert pool and the type-specific alert pools. An agent identifier is also stored to map the universal alert entry to the type-specific alert entry.

For each evaluation agent, the broker creates a record in an *alert status* relation for every alert. This record is a quadruple of *alert identifier*, *evaluation agent identifier*, a floating-point *evaluation value* and a *state value*. The evaluation value is the result of an *assertion* message from the corresponding evaluation agent for the given alert identifier. In the prototype, a value of 1.0 indicates a valid alert and a value of -1.0 indicates an invalid alert. A value of zero implies that the evaluation agent did not have enough information to validate or invalidate the alert. A null evaluation value implies that an agent has not yet processed the alert.

The alert status relation requires one entry for each alert for every agent deployed in the framework. If there are n alerts and m agents then there are $n \bullet m$ entries in the alert status relation. Although this relation can become very large, the simple design and query power of the relation's structure make it the preferred storage method in our implementation.

4.3.2. Alert state

Alert state is stored in the state value of the alert status relation as shown in table 4.4. The default value of an alert state is "available" unless the agent is dependent on other agents, in which case the default state is "conditional". The broker will change the state entry from "conditional" to "available" when all of the agent's dependencies have been met for the alert. The broker sets the state to "in progress" when a *next* message is sent by the evaluation agent and returned successfully to the agent. The state is changed from "in progress" to "processed" when the broker receives the *complete* message for this alert from the evaluation agent.

Table 4.4. Alert state values

Value	Alert State
1	*in progress*
2	*processed*
3	*conditional*
4	*available*

4.3.3. Agent dependencies

Agent dependencies are stored in a *dependency* relation that maps an agent, or the *dependent agent*, to the agent on which it depends, or the *parent agent*. The relation also stores the dependency condition as shown in table 4.5. The dependent agent can have multiple parent agents and can also be a parent agent so long as the mapping remains acyclic, forming a dependency tree. Agents without entries in the dependency relation both as the dependent agent and the parent agent are autonomous.

Table 4.5. Alert dependency conditions

Value	Condition
1	Evaluation > 0
2	Evaluation < 0

When an alert state is changed from "in progress" to "processed" as the results of the *complete* message, the alert broker determines if the completion of the alert has caused any dependencies to be satisfied. The broker first determines which agents depend on the agent that sent the *complete* message. Then for each dependent agent, the broker checks that all the dependencies have been satisfied. If so, the broker changes the state from "conditional" to "available" to allow dependent agents to process the alert.

Through the use of dependencies, the broker provides hierarchical processing capabilities. When an evaluation sends a *next* request, the broker queries the alert database to determine the next alert to process. To generate the list of alerts ready for processing, the alert broker selects all records in the alert status relation with a state value of "available". The list is ordered by descending alert timestamp in our implementation. This ordering has the potential for starvation under heavy load if the evaluation agents cannot keep up with the

number of incoming alerts. The benefit of this ordering is that it favors newer alerts, making the system timelier.

4.3.4. EMMAD alert verification

The EMAAD IDS developed and used by LANL's network security team detects unusual network activity patterns from network flow records by comparing average host activity to current activity patterns. An alert is generated when the level of anomalous activity exceeds a threshold. The system is used extensively by security analysts and has proven to be highly useful in practice.

After several months of use, a set of anomalies were identified that triggered because of hosts occasionally running printer discover applications. These applications search for printers on the internal network by scanning with port 161 UDP traffic in the network flow data. This traffic pattern is known to be benign, but is still anomalous by definition in EMAAD.

Instead of making changes to EMAAD, an alert verification agent was attached to the prototype EMAAD alert broker to check for large occurrences of port 161 UDP traffic in a short period of time. If the number of port 161 UDP flow records exceeds a threshold at the time of an alert, the agent sends an *assertion* message with a value of -1.0 indicating that the alert is a false positive. If a large number of 161 UDP records are not found, a value of 1.0 is returned to indicate a valid alert. The zero value is returned if the log files are not available, which can occur because processing delays.

This approach is preferable to making changes to the EMAAD system as there is a possibility that an attacker could use knowledge of the agent's design to scan the LANL network on port UDP port 161. By changing EMAAD to ignore this activity, it would be blind to all such attacks in the future. The agent-based approach allows users to filter out 161 UDP scans generated by EMAAD but analysts can still audit this activity when necessary. Also, the agent-based approach is useful for threshold tuning. The agent's threshold defining what constitutes a port 161 UDP scan is very subjective. By tuning the threshold using an agent, a value can be selected without losing alerts. When changing the threshold, the agent simply re-processes all the alerts to test the new threshold setting.

A web interface was developed to allow security analysts to view alerts in the EMAAD alert broker's database. The interface's default operation displays alerts that have a value of 1.0 for the alert verification agent, thus filtering out known false positives. The analyst can also view alerts with values of zero or -1.0 if needed.

4.3.5. Snort alert verification

Application-level TCP/IP attacks require a fully established connection between the attacker and the target hosts. Establishing a connection requires a three-way handshake consisting of three packets. Additionally, the attack packet must be sent requiring at least one additional packet. By locating a network connection event between the source and destination IP addresses over the port combination in the Snort alert, it can be determined whether or not a successful connection was established. The absence of a successful connection indicates that something (e.g. a firewall or IPS) prevented the attack from occurring, even though Snort detected the attack packet due to its placement on network.

To verify that a successful connection was established for TCP alerts, an alert verification and event correlation agent was attached to the Snort alert. The agent first correlates network connection records generated by LANL's proprietary NetHead IDS to the Snort alert. If a TCP connection record with more than three packets is located spanning the timestamp of the alert and matching the source and destination IP and source and destination ports, an *assertion* message with an evaluation value of 1.0 is returned indicating that a successful connection was established. The correlated event is also returned as XML in an *event* message to provide the analyst with additional information. If no record exists matching the above criteria, a value of -1.0 is returned indicating that no connection was established between the hosts. A value of zero is returned if the alert is not for a TCP attack or if the network connection logs are unavailable for the necessary time period, a possible occurrence due to unreliable data.

As with the EMAAD alert broker, a web interface has been implemented to display Snort alerts that have been validated by the evaluation agent. The interface displays the Snort alert information along with the correlated NetHead event, providing the user with additional

information. Security analysts also have the ability to display invalidated or unprocessed alerts.

4.3.6. Multi-paradigm event correlation

To analyze an IDS alert, a security analyst requires information from additional data sources to correctly make a determination of the alert's importance. We deployed several event correlation agents within our prototype framework to automate the task of event correlation. Additionally, we utilize correlated events in our research on multi-paradigm alert correlation described in chapter 5. Correlated events are transformed into feature vectors that represent an abstract "fingerprint" of a host at the time of the alert. By performing correlation using the correlated events, alerts from multiple IDS sensors and paradigms can be correlated. The event correlation framework assists in the first phase of the correlation process, by automating the correlation of events from multiple heterogeneous datasets.

Individual agents were designed to correlate events from Snort, TippingPoint and NetHead. For each type, a time window spanning the alert timestamp is defined as shown in table 4.6. The search criteria used to locate correlated events is the IP address of the host on the internal network and the timestamp within the time window and is selected as it is common to the three alert types that are supported by the prototype system: Snort, EMAAD and TippingPoint. More complex event correlation agents can be constructed, however this information is of highest importance to our work on multi-paradigm alert correlation.

Table 4.6. Event correlation time windows

Type	Window (hours)
Snort	24
TippingPoint	12
NetHead	1

A web interface was created for security analysts to view alerts from EMAAD, Snort and TippingPoint along with correlated events. Without the event correlation framework, analysts must perform the correlation operations by hand, often executing multiple queries

over multiple interfaces. The framework is used to automate these common tasks, making the analyst more effective and efficient in detecting and responding to intrusions.

4.4. Summary

To summarize, we present a framework for alert verification and event correlation that addresses limitations in prior work in alert correlation, most notably ease of deployment, flexibility and scalability. The framework is easy to deploy due to its use of autonomous agents and web services for communication between elements of the system. The system is scalable in that agents can be run asynchronously on multiple hosts. Our framework contributes to the IDS research by addressing these issues and enabling future research in the area of alert correlation.

Additionally, we describe the prototype implementation of the framework and several agents implemented to perform alert verification and event correlation at LANL. The prototype implementation supports alerts from three IDSs from varying paradigms. This shows the flexibility of our framework in practice. The system is also distributed and uses HTTP for communication, allowing the framework to be deployed in a variety of network environments.

The multi-paradigm event correlation agent implementation is utilized in chapter 5 to facilitate the prototype multi-paradigm alert correlation algorithm implementation. The event correlation agents are used to gather log records that are converted into feature vectors used in the correlation algorithm. By leveraging the event correlation framework and the web service framework for the alert correlation approach in the following chapter, we continue to build upon the concepts presented so far in this dissertation.

5. Multi-Paradigm Alert Correlation

To monitor and protect networks from evolving threats, network security administrators deploy multiple types of intrusion detection systems. Alert correlation is needed to group related alerts to reduce the number of individual alerts and discover relationships amongst alerts. Current approaches to alert correlation require a normalization phase prior to correlation to map alerts into a common representation and cannot easily be generalized to correlate alerts from disparate detection paradigms.

To overcome this problem, we present a generalized, multi-paradigm alert correlation approach using feature vectors derived from network log datasets. This differs from current approaches that correlate alerts using fields in the alert schema or a set of expert rules.

Domain experts evaluated our prototype implementation using unlabeled data and alerts collected from the Los Alamos National Laboratory (LANL) network. Initial results show that our system outperformed a control algorithm that correlates alerts using fields in the Snort schema.

5.1. Introduction

Network attacks continue to evolve in complexity, necessitating the deployment of multiple types of intrusion detection and prevention systems. Deploying a lone signature-based intrusion detection system (IDS) is no longer adequate for most network environments. Anomaly detection systems (ADSs) are needed to detect unknown attacks. Intrusion prevention systems (IPSs) are capable of defending against threats by taking action such as terminating a network connection. As organizations add systems to monitor and protect their networks and host systems, alert correlation techniques are required to discover relationships between sensors and detect sophisticated multi-step, distributed attacks.

Alert correlation techniques are used to streamline network and computer security analysis workflows by decreasing the number of alerts presented to the user and provide insight into complex network attacks. Low-level, or elementary, alerts are grouped together into clusters of related alerts. For instance a host scanning the Internet may generate

thousands of elementary alerts that can be correlated and presented to the user as a single event. Alert correlation techniques assist analysts by providing lower numbers of alerts and discovering patterns in the data that would otherwise go unnoticed.

Correlating amongst IDSs is difficult in that translation of the alert data into a unified schema is required. This process, known as *correlation*, is the first step of many current alert correlation techniques. Prior to correlation, alerts from heterogeneous sensors are mapped to a common schema expected by the correlation algorithm. In practice, normalization is challenging due to resource constraints and incomplete or missing information. More importantly, techniques relying on normalization do not allow for correlation between disparate detection paradigms. Correlating alerts amongst differing paradigms is a challenge for current techniques as they rely on the greatest common denominator of overlapping fields to perform correlation. This is typically a small number of fields, in some cases simply an IP address and a timestamp, timestamp as with many host-centric anomaly detection systems. With the limited amount of information provided in the alert schema, it is difficult to perform interesting or useful correlation when dealing with alerts from differing paradigms.

The majority of alert correlation research projects, open source projects and commercial applications, including expensive, complex security information management (SIM) solutions, rely extensively on expert rules and heuristics to perform alert correlation. This becomes a burden on analysts who must constantly develop and tune correlation rules to ensure accuracy. Maintaining this rule set requires deep domain knowledge and is susceptible to human error.

To overcome the drawbacks and limitations in current alert correlation techniques, we propose an approach that uses network and system log data along with IDS alerts to build a generalized alert profile that is used to correlate alerts from multiple IDS paradigms. We consider *events* to be records from information sources deemed relevant to an individual IDS alert.

Our approach involves building a statistical fingerprint for each IDS alert from network logs related to the internal, protected host at the time of the alert. A feature vector is built with features encoded as histograms and similarity between alerts is computed by comparing the distances between histograms. Similar alerts are placed in clusters that represent an

abstract correlation between the clustered alerts' network activity profiles. Using this technique, alerts from multiple IDS, IPS and ADS sensors can be correlated without the need of normalization, the use of an alert ontology or expert rules. Additionally, our approach does not temporally constrain the correlation process, allowing for long-term trend analysis and knowledge discovery. Although our current focus is on network datasets, our approach can be generalized to support data from host-based tools.

To evaluate our technique, we implemented a prototype offline correlation system and generated Snort alert correlation results from non-overlapping, independent one-month and six-month datasets. Security analysts at Los Alamos National Laboratory (LANL) evaluated the one-month dataset. The domain experts were asked to compare the correlation of our technique to that of a control group, which performed correlation by grouping alerts by fields in the alert schema. In our test, the experts preferred our technique to the control group, showing that our generalized correlation technique produces results that are better than an algorithm that groups alerts using the alert schema, as typically done in practice. Additionally, the experts showed a preference towards correlation results utilizing network log data in addition to the information in the alert data set. This shows that adding suitable log data yields improved results, an important factor in expanding our prototype in future study to incorporate additional alert databases for correlation. Lastly, we explore the average entropy of alert clusters in relation to the number of clusters generated by the clustering algorithm and use this metric to tune our results.

5.2. Alert Profile Construction

To begin the alert correlation process, related network log records, referred to as *events*, are gathered for each alert in the database. These events are used to construct feature vectors in the second phase of the process. Defining how events are related to a particular alert is challenging and is dependent on domain knowledge and available datasets. This is a critical aspect of the design and implementation of the system.

For our prototype, events are gathered from three datasets: Snort alerts, TippingPoint block logs and network connection records from proprietary LANL software known as NetHead. These three datasets are related in that they monitor network traffic and do so in

the same location on the LANL network as shown in figure 5.1. The TippingPoint sensor is an inline IPS while Snort and NetHead passively monitor for malicious activity via optical network taps. TippingPoint and Snort generate alert records and NetHead records network flow records.

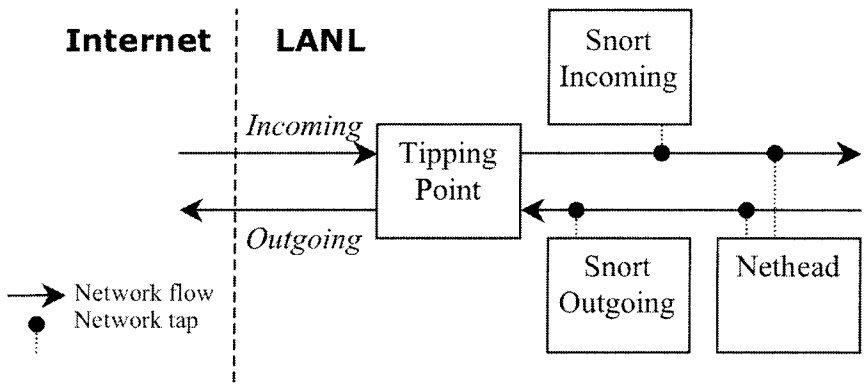

Figure 5.1. IDS sensor placement

These datasets were chosen for use in our prototype largely due to overlapping fields in the event records. Other datasets were considered but were ruled out as being noisy or insignificant. In hindsight, the TippingPoint data proved to be insignificant in that only 2000 related records were found out of a total of one million correlated events. In our future work we plan to develop techniques to guide selection of suitable datasets.

The three chosen datasets contain information on IP protocol, source and destination IP addresses and source and destination ports. This overlap is desirable, although not necessary, when building feature vectors in the second step of the correlation process. These fields are common to many network logs and IDS alert schemas and provide an ideal "common ground" for generating network traffic profiles of related events.

To identify events related to a given alert, the internal IP address and the detection timestamp of the alert are used to determine the search criteria. It is reasonable to assume that these fields will be available for any type of IDS system, including host-based systems. Given the placement of the sensors used in our prototype, each log record contains a single internal IP address and a single external IP address. This limits the detection view of our system to internal-to-external and external-to-internal network traffic patterns.

After establishing the search criteria for related events, the logs pertaining to the internal IP address are gathered for a time period related to the detection timestamp. A time window is established for each data set, depending on volume. Large datasets required a smaller window due to resource constraints in our test environment, although this can be reexamined in a production environment. Table 5.1 shows the data set volumes and corresponding window sizes for each type.

Table 5.1. Alert profile datasets

Type	Volume	Window
Snort	100+/day	24 hours
TippingPoint	1000+/day	12 hours
NetHead	3,000,000+/day	1 hour

For the time window, records that contained the internal IP address in the source or destination address field were stored in a relational database. The resulting records represent a profile of the network activity of the internal host before and after the alert. Ultimately, correlation results are dependent on the accuracy, reliability and information value of the log files used to construct the alert profile.

5.3. Feature Vector Generation

After building an alert profile, the information in the logs is abstracted into a feature vector that is used to perform correlation amongst alerts. The feature vector is used in the correlation phase as a "fingerprint" for the alert based on the network activity related to the alert. Each field in the feature vector is a histogram of network traffic information. Appendix D describes each feature in detail. The choice of fields is dependent on domain knowledge and available datasets as in profile construction. For instance, in our prototype the only time related feature is a duration histogram. An additional time feature can be added to incorporate time of day information into the feature vector with categories representing the hour of the day the events occurred. This feature would add a temporal dimension to the correlation results.

To illustrate an example of histogram computation, consider the log information for internal host 10.0.0.1 presented in table 5.2. Upon initialization, each feature contains a histogram with bin values of zero. For each related event, the bin count is incremented when a field value falls within the bin category or range. In this case, a protocol histogram with bins for "TCP", "UDP" and "ICMP" is encoded as [3, 1, 0]. A duration feature with bins for "short", "medium" and "long" can be encoded as [2, 1, 1]. The logs presented in table 5.3 differ considerably and are encoded [0, 0, 3] for protocol and [3, 0, 0] for duration.

Table 5.2. Logs for alert 1 with internal IP 10.0.0.1

Source IP:Port	Destination IP:Port	Protocol	Duration
10.0.0.1:3576	1.2.3.4:8080	TCP	2
10.0.0.1:161	9.8.7.6:161	UDP	0
1.2.3.4:1234	10.0.0.1:443	TCP	0
1.2.3.4:2233	10.0.0.1:22	TCP	1000

Table 5.3. Logs for alert 2 with internal IP 10.0.0.2

Source IP:Port	Destination IP:Port	Protocol	Duration
10.0.0.2:0	192.168.1.1:0	ICMP	0
10.0.0.2:0	192.168.1.2:0	ICMP	0
10.0.0.2:0	192.168.1.3:0	ICMP	0

The use of histograms highlights the differences in the example alert profiles, but the histograms must be scaled to account for log volume before they can be compared in the correlation phase. A count is kept for each feature and incremented whenever a bin value is incremented. Not every log will contain the fields relevant to a feature and are ignored in the context of the missing field, but still considered for other applicable features. For instance, bytes values are used to generate the vector in our prototype but Snort and TippingPoint are packet-based systems and therefore do not have information on byte counts in their logs.

After processing all the correlated events, each bin count is divided by the total number of log records yielding a value between 0.0 and 1.0. In the above example, the first protocol encoding is scaled to [0.75, 0.25, 0.00] and the second protocol encoding is scaled to [0.00,

0.00, 1.00]. Table 5.4 shows the histogram representations of the protocol and duration features for alerts 1 and 2.

Table 5.4. Histogram examples

Alert	Protocol	Duration
1	[0.75, 0.25, 0.00]	[0.50, 0.25, 0.25]
2	[0.00, 0.00, 1.00]	[1.00, 0.00, 0.00]

5.4. Correlation

The final step in the process is to compute similarity between feature vectors to correlate alerts with similar network traffic signatures. A number of statistical or data mining techniques can be used to compute similarity and correlate feature vectors. However as a proof of concept, a simple distance measure is used in our prototype to compare histograms. Bray-Curtis distance [36], also known as Sorensen distance and widely used in Ecology, was selected as it is normalized making threshold selection easier than using a distance measurement like Euclidian distance, which requires that distances be scaled. A distance of 0.0 indicates an exact match, with higher values indicating that the histograms are different. A distance of 1.0 indicates maximum dissimilarity.

$$d(A,B) = \frac{\sum_{i=1}^{n} |A_i - B_i|}{\sum_{i=1}^{n} (A_i + B_i)} \quad (1)$$

$$D(X,Y) = \frac{\sum_{i=1}^{n} d(X_i,Y_i) \times w_i}{\sum_{i=1}^{n} w_i} \quad (2)$$

Bray-Curtis distance is computed as shown in equation (1) where $d(A,B)$ is the distance between histograms A and B. Given the example data in table 5.4, the distance for the protocol histograms is 1.00 and 0.50 for the duration histograms. To compute the distance

between feature vectors *X* and *Y*, equation (2) is applied. In our prototype, each feature is weighted with a value of 1, giving an overall distance of 0.75 for the two alerts, showing that they are highly unrelated.

The clustering algorithm shown in figure 5.2\ takes as input a minimum distance threshold, *t*, the set of alerts represented as feature vectors, *V*, and an empty set of alert clusters, *C*. For each feature vector, the distance between all other vectors is computed. If there is an alert that is less than *t* from the alert, the two alerts are places in a cluster, the centroid is computed and the cluster is added to *C*. If there is more than one alert whose distance is less than *t*, the closest alert is chosen. If no vector in *V* is within *t*, the distance between the alert and all clusters in *C* is computed. The alert is added to the closest cluster with distance less than *t* from the alert. If no such cluster is found, the alert is removed from *V* and placed as a singleton cluster in *C*. This process continues until there are no alerts in the database.

```
Let t be a value between 0.0 and 1.0
Let V be the set of feature vectors
Let C be the set of clusters, initially empty

For feature vectors Vx in V, i ← 1 to size(V):
  For clusters Cx in C, j ← 1 to size(C):
    Compute D(Vi, Cj)
    Find min distance less than t, if any, and add Vi to Cj
  If Vi not clustered:
    For feature vectors in V, j ← i to size(V):
      Compute D(Vi, Vj)
      Find min distance less than t, if any:
        Create new cluster c containing Vi, Vj
        Add c to C
        Remove Vi, Vj from V
      If Vi not clustered:
        Create new cluster c containing Vi, add to C
        Remove Vi from V
```

Figure 5.2. Clustering algorithm pseudocode

5.5. Results

In our evaluation we show that adding suitable datasets to the correlation process results in improved results according to domain experts evaluating a random sample of alerts. We also show that our generalized technique is comparable in terms of user preference with techniques used in practice for correlating IDS alerts. These techniques correlate alerts according to features in the alert schema and cannot be generalized for multi-paradigm correlation without normalization.

The evaluation dataset contains log records and IDS alerts collected on the LANL network. The dataset is unlabeled and therefore we cannot show accuracy in terms of false positive and false negative rates as typically presented in alert correlation research. Instead we rely on domain experts to evaluate a sample of Snort alerts. We feel these results, although subjective, are applicable to real-world environments because the data set was collected from a live network and evaluated by domain experts, unlike many results that are generated from small, artificial data.

In our evaluation, security analysts from LANL were asked to compare correlation results from one month of Snort alerts correlated using four different approaches: a control group clustering on fields in the Snort alert schema and three instances of our algorithm with varying datasets used to generate feature vectors. For each Snort alert, correlated events were gathered from Snort, TippingPoint and NetHead to build the network traffic profile and feature vector. Next, a random sample of these alerts was selected for evaluation. The sample of alerts consisted of up to three alerts for each distinct signature for the month for a total of 26 unique alerts.

The domain experts were asked to rank each cluster containing the sampled alert on a scale of 1 to 10 for each of the four techniques, with low values indicating inaccurate or uninteresting clustering and high values indicating well-correlated or interesting results. For example, an alert for a web attack in a cluster of email alerts indicates that the web attack is poorly correlated. Conversely, a large cluster of similar web attacks would be of interest to the analyst. This metric was left to the analysts' discretion and expert knowledge. Analysts did not know which approach was used to generate the correlated clusters.

The first correlation technique, the control algorithm, clusters alerts in a relational database by using a series of relational GROUP BY operations. There are five iterations of the algorithm using the field values in table 5.5 to merge alerts into clusters. In each iteration, the set of distinct tuples matching the group by fields is computed along with a count of merged alerts in the tuple. If more than one alert is merged in the tuple, the matching alerts are placed in a new cluster and removed from the database. If not, the alert is left in the database for consideration in the next iteration. After all iterations are run, each alert remaining in the database is placed in a singleton cluster.

Table 5.5. Fields used by control clustering algorithm

Iteration	Group By Fields
1	Signature, Src IP, Dst IP, Src Port, Dst Port
2	Signature, Src IP, Dst IP, Dst Port
3	Signature, Src IP, Dst IP, Dst Port
4	Signature, Src IP, Src Port, Dst Port
5	Signature, Dst IP, Src Port, Dst Port

Table 5.6. Evaluation algorithms

Algorithm	Description	Datasets
1	Control	Snort
2	Snort alerts only	Snort
3	IDS/IPS alerts	Snort, TippingPoint
4	All datasets	Snort, TippingPoint, NetHead

The three remaining techniques apply our approach with incremental input sources, as shown in table 5.6, used in generating the network activity profile. The similarity threshold used in this evaluation was 0.2. Our method for selecting this threshold value is presented in section 5.6. The evaluation results for the random sample of alerts show that algorithm 4 was the preferred technique according to the expert opinion of the analysts given the sample alerts and the available input datasets. Table 5.7 shows the average alert score and standard deviation for each algorithm.

Table 5.7. Domain expert evaluation results

Algorithm	Ave. Score	Std. Deviation
1	5.04	1.99
2	4.56	1.70
3	4.56	1.90
4	5.57	1.65

These results support our claim that the generalized algorithm using multiple datasets competes with approaches clustering on fields in the alert schema. Furthermore, the control group results generated 4476 clusters over a six-month period, which included the time period of the sampled alerts, compared to only 931 clusters when using algorithm 4 with a threshold of 0.2. This shows considerable reduction in the number clusters presented to the analysts while at the same time providing interesting clustering of the sampled alerts.

Also, among the three algorithms using the generalized technique, the algorithm building the feature vectors with all three datasets was preferred over the others. This supports our claim that adding suitable data in the alert profile generation phase will result in improved correlation. It was also apparent that the small TippingPoint dataset did not have a noticeable impact on the overall results. A similar evaluation can be used in practice to determine if a dataset should be included in alert profile construction. In this case, the results show that the correlated TippingPoint events can be removed from the alert profiles.

The argument can be made that correlating events using multiple data sources can introduce information into the correlation process that leads to inaccurate or misleading results. By incrementally selecting datasets as shown in table 6, we show that adding more event records resulted in improved correlation according to the preferences of our domain experts.

5.6. Threshold Tuning

When evaluating the domain expert's cluster rankings, it became apparent that the amount of entropy in the clustered Snort alerts was related to the expert's rankings. Clusters with low entropy equate to high rankings while clusters with high entropy equate to low rankings. This observation is used to tune the clustering algorithm similarity threshold. If the

threshold is low, there are too many clusters for the analysts to examine. If the threshold is high, there are fewer clusters but the entropy increases, making analysis difficult.

To tune the threshold, the clustering algorithm is run with similarity thresholds ranging between 0.0 and 0.5 in increments of 0.05. For each increment, the cluster entropy is computed for each cluster and the average is calculated.

$$entropy(C) = \sum_{i=1}^{N}\left(-1 * \sum_{j=1}^{Mi} p_j \log_2 p_j\right) \quad (3)$$

Equation 3 calculates the entropy of cluster C where N is the number of features. Mi is the number of distinct items for feature i and p_j is the probability that feature i's value is found in cluster C. For each cluster of Snort alerts, entropy is calculated and summed for five features: source and destination IP, source and destination port and signature name.

The cluster size and cluster entropy from six months of network data is shown in figures 5.3 and 5.4. The ideal thresholds fall in the shaded regions where entropy and cluster volume are low. The shaded boxes and the values in table 5.8 represent the acceptable similarity threshold values for the evaluation data set given the resources at LANL. If the threshold is lowered, there are too many clusters to be analyzed each day. If the threshold is raised, the entropy values become high, indicating noisy clusters. The ideal threshold to select is one that has the lowest overall entropy that can be managed given the personnel resources dedicated to analysis of the data.

Table 5.8. Selected similarity thresholds

Threshold	# Clusters	Clusters/day	Ave. Entropy
0.15	1187	7	0.27
0.20	734	4	0.38
0.25	423	2	0.57

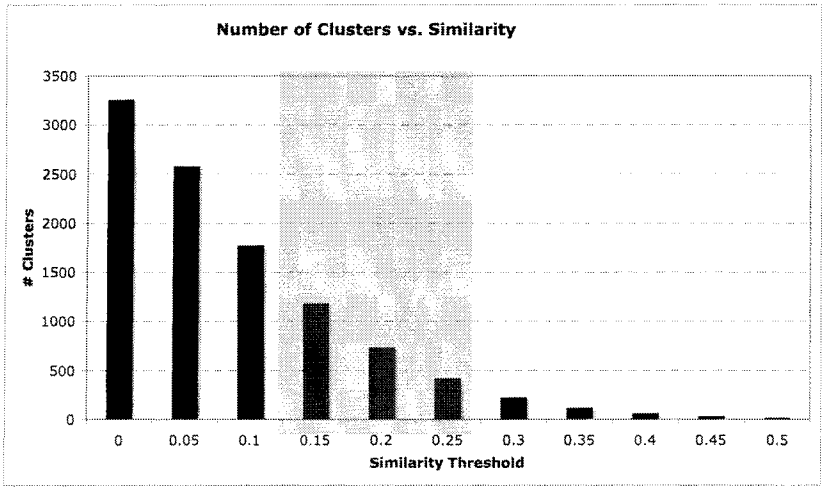

Figure 5.3. Cluster volume related to the similarity threshold

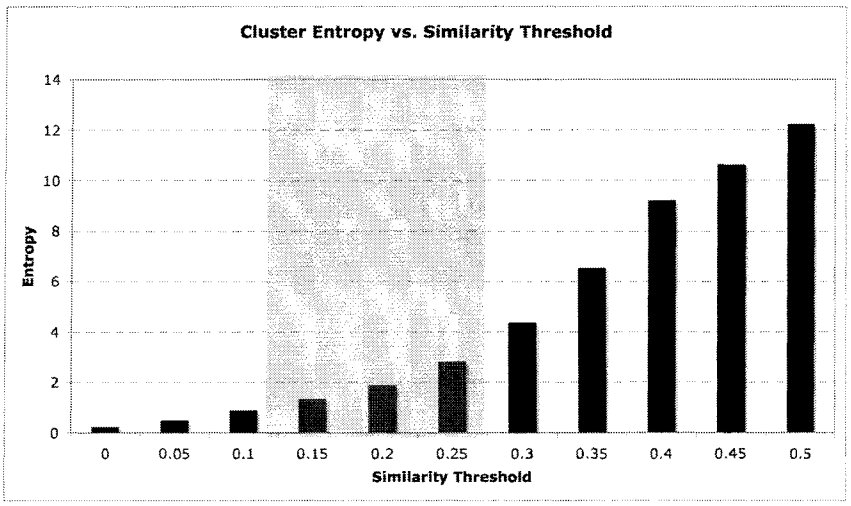

Figure 5.4. Average cluster entropy related to similarity threshold

5.7. Discussion

Several interesting results were discovered in evaluating a second result set of six months of Snort alerts and correlated events. The signature database contained a combination of published and custom Snort rules. Several of the signatures are now deprecated because of high false positive rates. In one case a rule looking for the string "nc%20", meant to detect remote NetCat execution, was known to trigger false positives regularly on a web server hosting the periodic table of the elements. When the page was viewed the rule triggered because the element "zinc" appeared in the page followed by the "%20" HTML encoding for

a space character. In the correlation results, alerts from this web server are grouped together in a large cluster that included alerts spanning the six-month time period. Alerts in this cluster differentiate themselves from other alert instances matching the signature due to the unique traffic pattern generated by the web server.

Another interesting example involved a custom rule looking for a byte sequence associated with a known rootkit. The rule triggered consistently on web servers downloading JPEG images that contained the byte sequence by chance. However, one particular set of correlated alerts was generated from a single internal host over a period of ten minutes. This cluster was very different from the normal pattern and showed that our approach is capable of finding patterns that otherwise go unnoticed.

This signature is also of interest in that the name of the rule changed at a point in time late in the six-month period. In many instances, alerts with both signature names were grouped in clusters. This illustrates the correlation algorithm's ability to group alerts without the need of normalization. The traffic patterns of these alert instances were the same, so they were correlated regardless of signature name.

Although this evidence is anecdotal, our observations show that our alert correlation technique is a useful tool for data exploration. Interesting patterns and trends are discovered that would otherwise go unnoticed.

5.8. Summary

In summary, we present an alert correlation technique that is applicable to multiple intrusion detection paradigms. Our correlation approach is dependent on network events related to IDS alerts, not the features in the alert schema. By making our correlation criteria independent of the alert schema our approach can be applied to alerts from any number of IDSs. Another benefit of this approach is that it does not rely on an alert ontology or normalization to correlate alerts. We feel this to be an important contribution as normalization and maintenance of an alert ontology are resource intensive tasks in real-world environments where multiple IDS, IPS and ADS sensors are deployed.

Another contribution of this work is its evaluation and tuning using live data from the LANL network. We show through domain experts that our technique provides interesting

and useful results compared to the available alternatives. We also show that in our evaluation of real-world data that more input datasets result in improved correlation results, according to the domain experts. Lastly, we provide guidelines for threshold tuning by weighing cluster volume against cluster entropy. These results are important in that the techniques utilized in evaluation and tuning can be utilized by organizations when to deploying an alert correlation system following our approach.

6. A Multiple Criteria Hill Climbing Algorithm for Alert Correlation Feature Selection

Feature selection is a common data-mining technique used to reduce the feature space of high-dimensional datasets. Typically, feature selection algorithms required labeled datasets to determine accuracy. As labeled networking and intrusion detection systems data is difficult to obtain in real world environments, applying feature selection is difficult in practice.

To address this problem, we present a feature selection for correlation of intrusion detection system (IDS) alerts that is optimized to run in production environments where labeled data is not available. Our approach locally optimizes a set of features using hill climbing based on any number of user-defined evaluation criteria. In our prototype, we use cluster entropy and cluster volume to optimize a feature set developed in our prior work on multi-paradigm alert correlation.

6.1. Introduction

Feature selection is a common step in many data-mining problems where it is desirable to compare the results of various feature sets to optimize results. Feature selection is important for high-dimensional datasets where performance of the data-mining algorithm is a critical factor. Reducing the number of feature is important in this case to optimize execution time. Feature selection is also used to increase the number of features. This is done to improve the accuracy of the algorithm by considering features not initially used in testing the data-mining algorithm.

Applying feature selection to IDS alert correlation is an interesting but difficult problem when working with real-world datasets. Feature selection techniques assume that the input dataset is labeled for classification. In the case of intrusion detection systems this is typically a label of "intrusive" or "normal". The feature selection algorithm then reports its classification accuracy [19]. In cases where only unlabeled datasets are available, such as production environments and large test networks, feature selection cannot be applied in this way, as each alert requires to be labeled with a classification. This process is time consuming

and prone to human error as it relies on domain experts to apply labels. An alternative is to establish new evaluation criteria that can be applied to real-world, unlabeled datasets to perform alert correlation feature selection.

We present an efficient feature selection algorithm for alert correlation that is designed for use in production environments where labeled data is not available, although our approach is applicable to datasets with labeled data as well. The algorithm present in this dissertation selects features based on the overall performance of one or more user-defined, weighted evaluation criteria. In executing the algorithm, an initial, or *candidate*, set of features is defined from a set of available features. An alert correlation algorithm is then run with the candidate features to establish a baseline for comparison in feature selection.

Feature selection begins with reverse feature selection, where features are removed from the candidate feature set one at a time and the correlation algorithm is applied to each reduced feature set. This process continues until the correlation results stop improving relative to the baseline performance of the candidate feature set. Next, forward feature selection is applied to the potentially reduced feature set. In this phase, features omitted from the candidate feature set are added one at a time from the larger set of features. Like in reverse feature selection, this process continues until the performance stops improving.

Our algorithm is efficient in that it continues feature selection only as long as the overall performance improves. We use a hill-climbing approach to optimize the feature set based on the candidate feature set as a starting point. This dramatically limits the feature space by only exploring paths with improved results making the algorithm ideal for production environments where tuning of correlation algorithms must be performed periodically and quickly.

In our prior work on multi-paradigm alert correlation, we showed that there is a tradeoff between the number of clusters and average cluster entropy when tuning our similarity threshold. We used these properties as evaluation criteria to tune the feature set in our alert correlation implementation described chapter 5. Results from executing our feature selection algorithm on our test data shows that our candidate feature set showed improved performance when a single feature was removed.

In this work we provide three contributions:

1) Evaluation criteria for comparing alert correlation results using cluster volume and per-cluster entropy.
2) An efficient hill-climbing algorithm for alert correlation feature selection suitable for production environments.
3) Performance results from three feature sets considered for use in our research on multi-paradigm alert correlation.

The organization of this chapter is as follows. Section 6.2 reviews our prior work on multi-paradigm alert correlation. Section 6.3 and its sub-sections describe the feature selection algorithm. Section 6.4 presents the results of applying our feature selection algorithm to three feature sets using our multi-paradigm alert correlation algorithm and section 5 summarizes the chapter.

6.2. Multi-Paradigm Alert Correlation

In our prior work we defined a multi-paradigm approach to IDS alert correlation that correlates alerts using network traffic profiles instead of fields in the alert schema or expert rules as is commonly done in practice. For each alert, related logs, called *correlated events*, are gathered matching the internal host associated with the alert at the time of the alert. These events are used to construct a network traffic profile of the internal host before and after the alert occurred. The logs are transformed into feature vectors of network features like IP addresses, ports, bytes counts, etc. These feature vectors are used to correlate IDS alerts by clustering alerts with a similarity function.

In the evaluation of our results, it was apparent that a low number of clusters combined with low per-cluster entropy were desirable. Maintaining a low number of clusters is important in that security analysts require tools that do not inundate them with more information than they can handle. Maintaining low entropy is important in that clusters with high entropy have very little information value and therefore do not provide the analyst with meaningful correlation.

Given these criteria, we tuned our similarity threshold to find a point where there exist an acceptable number of clusters generated per day with low overall entropy. Analyst resources and an estimation of the number of clusters a full-time analyst could manage in a single day determined the number of acceptable clusters. We then selected thresholds based on this number where the entropy was the lowest. As we are using Bray-Curtis distance measures [36] for our similarity measure, threshold values ranged from 0.0 to 1.0. Threshold tuning was run in increments of 0.05 although more precise tuning can be performed if necessary. This experiment showed that thresholds values of 0.15, 0.20 and 0.25 provided acceptable results although for the purposes of this chapter, a threshold of 0.20 is used to show the results of feature selection.

6.3. Feature Selection Algorithm

This section details the feature selection algorithm. Sections 6.3.1, 6.3.2 and 6.3.3 describe the evaluation, feature set comparison and feature selection algorithms respectively. Two functions are implementation dependent: the alert correlation algorithm and the evaluation algorithm. The alert correlation function is run by the evaluation function to build the correlated dataset. The evaluation function computes the evaluation criteria as defined by domain experts implementing the system.

An explanation of variable names used in the following sections is provided in table 6.1. Variables with uppercase letters are sets or arrays while lower case values are integers or scalars.

Additional explanation of the feature set notations is necessary. We identify a set of all possible obvious and abstract features called F_{total}. Obvious features are those present as fields in the log records of our dataset. Examples of obvious features are the signature identifier and the source IP address of a Snort alert. Abstract features are those created by combining features to create a new feature, for example combining bytes transferred and duration from a network connection record to create a byte rate feature.

Table 6.1. Variables used in feature selection algorithm

Variable	Description
F	A set of features to be used when computing the *Correlate* and *Evaluate* function
F'	Intermediate feature sets created during forward and reverse feature selection
F_{test}	A set of features to be considered for addition during forward feature selection
F_{all}	The set of all features defined for alert correlation
E, E_1, E_2	Arrays of scalars representing evaluation results
v	A scalar representing the performance of a feature set returned by the *Compare* function
v_{min}	A scalar used to store the lowest evaluation value found during feature selection
F_{min}	Feature set that corresponds to v_{min}
E_{min}	Evaluation criteria results that correspond to v_{min}
W	A set of weights for each evaluation criteria whose sum is 1.0

Due to the inclusion of abstract features, F_{total} could be unbounded. Therefore a finite set of features, F_{all}, is selected from F_{total} by the domain experts to be considered during feature selection. The selection of features for F_{all} is dependent on the design and implementation of the correlation algorithm. For instance, features common to a variety of network security datasets were included to make our approach IDS independent. IDS specific features like a signature identifier were omitted because they restricted our technique to a particular IDS.

As F_{all} is a potentially large number of features, some of which may not be relevant, a candidate set of features, F, is chosen from F_{all}, again relying on domain expertise in network security to select features. For example, in our candidate feature set, the feature "time of day" is omitted, even though this was an obvious feature in our dataset.

Three additional feature sets are described in table 6.1. The feature set F_{test} is used during forward feature selection to evaluate the performance of features in F_{all} that were not included in the candidate feature set, F. The feature set F' is used by the reverse and forward feature selection phases to store intermediate feature sets. Lastly, F_{min} is used to track the optimally performing feature set during the feature selection process. This feature set is outputted upon completion of execution.

6.3.1. Evaluation criteria

A key aspect of our feature selection approach is the selection of evaluation criteria. As shown in the following section, our algorithm allows for any number of weighted criteria to be considered during feature selection. Since our focus is on real-world datasets, no labeled data is available and therefore it is impossible to obtain results in terms classification accuracy. This necessitates the creation of alternative evaluation criteria to compare results during feature selection.

The *Evaluate* function computes the performance of a feature set based on the defined evaluation criteria and returns an array with an entry for each evaluation criteria. Choosing evaluation features is implementation dependent, although in our prior work we determined that the number of clusters and the per-cluster entropy of clusters are good indicators of alert correlation performance. Therefore, we define two factors, equally weighted in our implementation, to guide feature selection: the average cluster entropy and the number of clusters.

Input: F
Output: E

```
Evaluate (F)
  C = Correlate(F)
  entropy = 0.0
  count = size(C)
  for i = 1 to count
   entropy += CalculateEntropy(C[i])
  entropy = entropy / count
  E = [ count, entropy ]
  return E
```

Figure 6.1. Evaluation function computing cluster count and average entropy

The *Evaluate* function in figure 6.1 computes the number of clusters and the per-cluster entropy and returns an array of scalar values containing the results. The *Correlate* function is not shown for brevity, as it is dependent on implementation. The *Correlate* function used to generate results in section 6.4 implements the multi-paradigm alert correlation algorithm presented in chapter 5 to correlate Snort alerts. For the purposes of this chapter, the *Correlate* function returns a set of clusters, C, given a set of features, F.

The *CalculateEntropy* function is provided in appendix E and computes per-cluster entropy using five fields in the Snort alert schema: source and destination IP addresses, source and destination ports and signature name.

6.3.2. Comparing feature sets

To compare two feature sets, a *Compare* function is executed on the evaluation results obtained by calling the *Evaluate* function on the feature sets. There must be a weight value assigned to each evaluation criteria. Figure 6.2 shows the *Compare* function.

Input: E_1, E_2, W
Output: a scalar value; if output value >1 E_2has better performance than E_1; if output value <1 E_1has better performance than E_2; if output value is 0.0 then the performance is equal

```
Compare (E1, E2, W)
  sumE = 0.0
  for i = 1 to size(W)
   sumE += E1[i]/E2[i] * W[i]
  return sumE
```

Figure 6.2. Generalized weighted comparison of evaluation results

The function returns a scalar value where a value of 1.0 indicates no difference between the two feature sets. A value greater than 1.0 indicates the first set of evaluation results, E_1, does not perform as well as E_2. A value less than 1.0 indicates that E_1's performance is better than that of E_2 for the evaluation criteria. This then implies that if E_1 is better than E_2, then the feature set used to compute E_1 is preferable to the feature set used to compute E_2 according to evaluation criteria implemented in the *Evaluate* function.

6.3.3. Feature selection

In sections 6.3.1 and 6.3.2, our evaluation and comparison functions were defined. These functions are used in the feature selection functions presented in this section. The functions *ReverseFeatureSelect* and *ForwardFeatureSelect* are run in succession by the *FeatureSelect* algorithm. The reverse and forward selection processes are hill-climbing

algorithms in that they only continue to run as long as the performance of feature selection improves. The *ReverseFeatureSelect*, *ForwardFeatureSelect* and *FeatureSelect* algorithms are presented in figures 6.3, 6.4 and 6.5 respectively.

In reverse feature selection, features are removed from the input feature set one-at-a-time. If removing a feature provides equal or better performance, the algorithm is called recursively on the reduced feature set. This continues as long as performance improves or remains constant. When the algorithm completes, the feature set with the best performance is returned along with the performance improvement value and the evaluation results.

```
Input:    F, v, E, W
Output:   Fmin, vmin, Emin

ReverseFeatureSelect (F, v, E, W)
  Fmin = F
  vmin = v
  Emin = E
  for i = 1 to size(F)
   F' = F - F[i]
   E' = Evaluate(F')
   v = Compare(E', E, W)
   if (v <= vmin)
        Fmin, vmin, Emin = ReverseFeatureSelect (F', v, E', W)
      vmin = Compare(Emin, E, W) // Recompute vmin
  return  Fmin, vmin, Emin
```

Figure 6.3. Reverse hill-climbing feature selection

Forward feature selection is very similar to reverse feature selection, only features are added one-at-a-time from the F_{test} feature set. Also, feature selection continues only if adding the feature improves performance, as adding additional features that do not improve the result only adds more complexity in the correlation process.

The final algorithm, *FeatureSelect*, is our solution to feature selection for alert correlation. The algorithm begins by evaluating the candidate set of features, F, taken from a set of selected features defined by domain experts, F_{all}. Then reverse feature selection is performed on the candidate feature set, returning a potentially reduced set of features, F'. Forward feature selection is then performed on F' by adding member of F_{all} that are not included in the candidate feature set. The final feature set is returned following forward

feature selection. The candidate feature set will be returned if reverse and forward feature selection cannot improve on the evaluation results of the candidate.

```
Input:    F, Ftest, v, E, W
Output:   Fmin, vmin, Emin

ForwardFeatureSelect (F, Ftest, v, E, W)
  Fmin = F
  vmin = v
  Emin = E
  for i = 1 to size(Ftest)
   F' = F + Ftest[i]
   E' = Evaluate(F')
   v = Compare(E', E, W)
   if (v < vmin)
        Fmin, vmin, Emin =
          ForwardFeatureSelect (F', Ftest - Ftest[i], v, E', W)
        vmin = Compare(Emin, E, W) // Recompute vmin
  return Fmin, vmin, Emin
```

Figure 6.4. Forward hill-climbing feature selection

```
Input:    F, Fall, W
Output:   F'

FeatureSelect (F, Fall, W)
  E = Evaluate(F)
  F', v, E = ReverseFeatureSelect(F, 1, E, W)
  F', v, E = ForwardFeatureSelect(F', Fall - F, v, E, W)
  return F'
```

Figure 6.5. Hill-climbing feature selection for multiple evaluation criteria

6.4. Results

This section presents feature selection results for alert correlation data generated over a six months time period between January 1, 2005 and June 30, 2005. For each Snort alert generated during this period, a network traffic profile of the internal host was generated and converted into a feature vector. The correlation process calculates the similarity of feature

vectors using Bray-Curtis distance and places alerts in clusters if the distance is less than a threshold value. In our experiments a value of 0.20 was used as our results in chapter 5 showed this to be an acceptable threshold based on the number of clusters generated per day. We selected a threshold that was at the high end of our available resources in terms of cluster volume. By selecting a higher number of clusters, the per-cluster entropy was reduced, maximizing information value while at the same time presenting IDS alert information at a manageable rate.

A set of features was generated using networking and intrusion detection domain knowledge. The set of features is shown in table 6.2 and described in detail in appendix D. From this group of features, a candidate set of features was selected. Also an expanded feature set and a reduced feature set were selected for comparison against the candidate feature set.

Table 6.2. List of features included in each feature set used in testing

Feature Set	Name	Features
A	Candidate	srcip , dstip, srcport, dstport, duration, srcbytes, srcpkts, dstbytes, dstpkts, protocol, int-ext, res-eph
B	Expanded	srcip , dstip, srcport, dstport, duration, srcbytes, srcpkts, dstbytes, dstpkts, protocol, int-ext, res-eph, hourofday
C	Reduced	srcip, dstip, srcport, dstport, duration, protocol, byterate

As described in section 6.3.1, we chose the number of clusters and the per-cluster entropy as the evaluation criteria for feature selection. The per-cluster entropy was computed as the sum of the entropy of five fields from the Snort alert schema (source and destination IP addresses, source and destination ports and signature name) divided by the number of clusters.

As shown in table 6.3, the reduced feature set creates the fewest clusters but the per-cluster entropy is very high. The expanded feature set, which only adds a single feature to the candidate, has lower entropy than the candidate feature set and a larger number of clusters. These factors must be weighed against each other when selecting feature sets prior to feature selection. Domain knowledge and experience are therefore important factors in this process.

Table 6.3. Results for cluster volume (CV) and per-cluster entropy (PCE) comparing initial feature sets (Candidate, Expanded, Reduced) to feature sets following feature selection (Candidate', Expanded', Reduced')

Criteria	Candidate	Candidate'	Expanded	Expanded'	Reduced	Reduced'
CV	461	499	665	706	273	270
PCE	3.576	3.231	2.621	2.427	6.338	6.301

The following sub-sections detail the results of applying our feature selection approach to the test feature sets. Sections 6.4.1, 6.4.2 and 6.4.3 provide results for the candidate feature set, the expanded feature set and the reduced feature set, respectively.

Our performance metric is computed as the percentage performance of the initial feature set, with values over 100% indicating a performance penalty. Values less than 100% indicate that the net change in CV and PCE improved compared to the initial feature set. For this experiment, we used the strict criteria that only features that improve the entropy will be considered in the feature selection process. The tables that follow indicate performance results that did not improve the entropy with a value of "NI" instead of a percentage value indicating no improvement in entropy over the feature set being compared.

6.4.1. Candidate feature set results

The feature selection results for the candidate feature set (feature set *A*) are presented in tables 6.4, 6.5 and 6.6 and figures 6.6, 6.7 and 6.8. Reverse feature selection begins by removing features one at a time from the candidate feature set as shown in table 6.2. Feature set *A.1* is the candidate feature set while features *A.2* to *A.13* are the reduced feature sets. Performance is presented relative to feature set *A.1*.

As shown in table 6.4, removing the *int-ext*, *protocol* and *srcport* features all improved the correlation results as per the evaluation criteria with *protocol* being the best performing. As per the algorithm, *ReverseFeatureSelect* is called recursively on each of these reduced feature sets. Only results of following the *protocol* path through the algorithm are shown in this section, as this feature set ultimately leads to the best results. For completeness, the feature selection results for removing the *int-ext* and *dstport* features are presented in appendix F.

Table 6.4. Results from the first iteration of reverse feature selection where a single feature is removed from the candidate feature set

Set #	Feature Removed	# Clusters	Entropy	Performance (%)
A.1	N/A	461	3.576	100.000
A.2	res-eph	426	3.773	NI
A.3	int-ext	475	3.443	99.330
A.4	protocol	499	3.231	98.598
A.5	dstpkts	455	3.700	NI
A.6	dstbytes	448	3.762	NI
A.7	srcpkts	450	3.801	NI
A.8	srcbytes	459	3.675	NI
A.9	duration	435	3.910	NI
A.10	dstport	476	3.465	100.158
A.11	srcport	488	3.325	98.841
A.12	dstip	442	3.920	NI
A.13	srcip	335	4.135	NI

Table 6.5. Results from the second round of reverse feature selection where a single feature is removed from feature set *A.4*; no feature set improved upon the results of feature set *A.4*

Set #	Features Removed	# Clusters	Entropy	Performance (%)
A.4	protocol	499	3.231	100.000
A.14	protocol, res-eph	459	3.495	NI
A.15	protocol, int-ext	529	3.037	100.619
A.16	protocol, dstpkts	487	3.486	NI
A.17	protocol, dstbytes	489	3.461	NI
A.18	protocol, srcpkts	496	3.404	NI
A.19	protocol, srcbytes	488	3.405	NI
A.20	protocol, duration	475	3.568	NI
A.21	protocol, dstport	528	3.040	100.484
A.22	protocol, srcport	536	2.988	100.657
A.23	protocol, dstip	486	3.497	NI
A.24	protocol, srcip	393	3.718	NI

The second round of reverse feature selection is shown in table 6.5. No improvement could be made over the candidate feature set with the *protocol* feature removed. In our implementation, only feature sets that improve or maintain the correlation performance are considered during reverse feature selection so in this case reverse feature selection ends.

Several features are very insignificant in this case (*int-ext*, *dstport* and *srcip*). The algorithm can be modified to remove features outside a significance threshold.

Forward feature selection, as shown in table 6.6, failed to improve upon the reduced feature set returned in the reverse feature selection process. The feature set returned is the candidate feature set with the *protocol* feature removed, yielding a modest 1.402% net performance gain.

Table 6.6. Results from forward feature selection; the performance from feature set *A.4* could not be improved

Set #	Feature Removed	Feature Added	# Clusters	Entropy	Performance (%)
A.4	protocol	N/A	499	3.231	100.000
A.25	protocol	hourofday	714	2.447	121.250
A.26	protocol	byterate	547	2.932	101.306
A.27	protocol	logtypes	476	3.494	NI

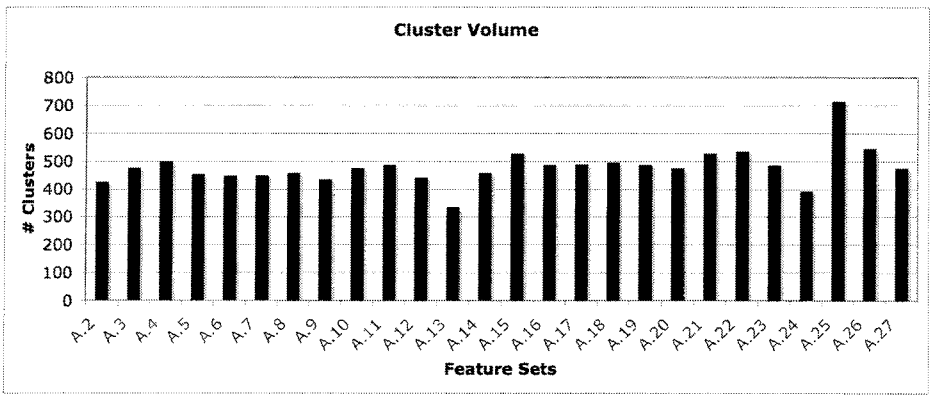

Figure 6.6. Cluster volume for candidate feature selection

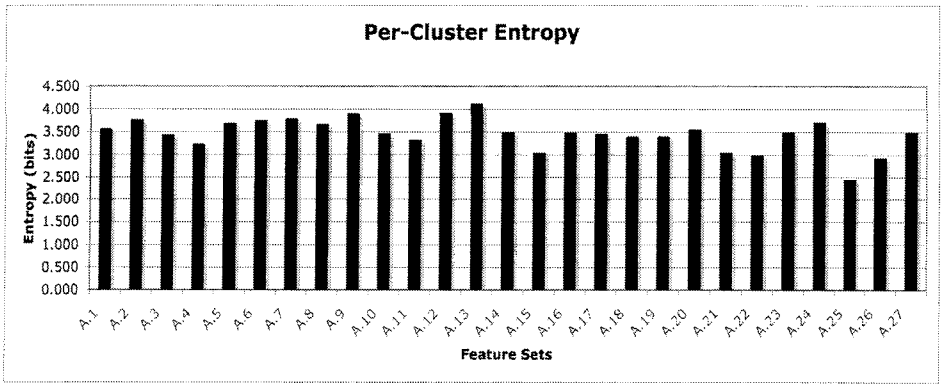

Figure 6.7. Per-cluster entropy for candidate feature selection

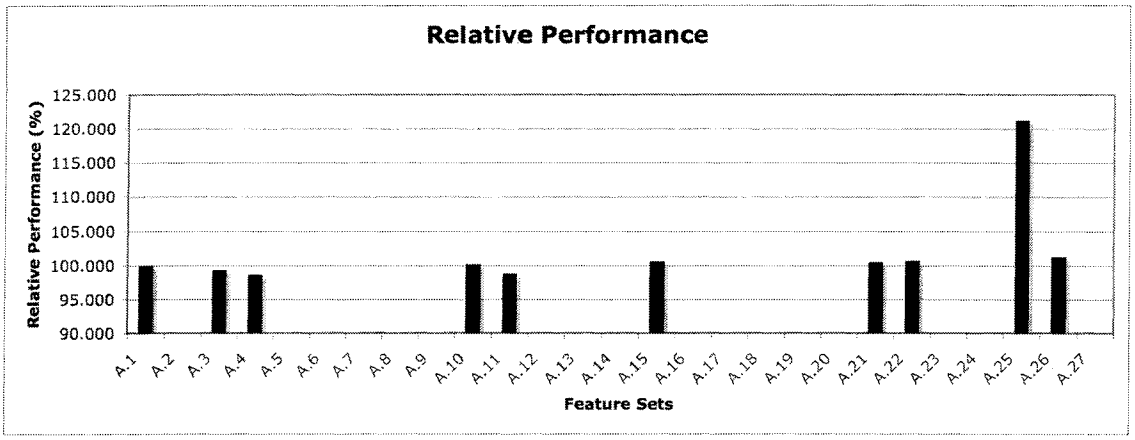

Figure 6.8. Relative performance for candidate feature selection; feature sets with empty values did not decrease the per-cluster entropy and therefore did not have a computed performance value

6.4.2. Extended feature set results

The feature selection results for the extended feature set are presented in tables 6.7, 6.8 and 6.9 and figures 6.9, 6.10 and 6.11. For this feature set, the only improvement over the initial feature set is obtained by removing the *int-ext* feature during the first iteration of reverse feature selection.

Table 6.7. Results from the first iteration of reverse feature selection where a single feature is removed from the extended feature set; feature set *B.1* is compared to feature sets *B.2* to *B.14*

Set #	Feature Removed	# Clusters	Entropy	Performance (%)
B.1	N/A	665	2.621	100.000
B.2	hourofday	461	3.576	NI
B.3	res-eph	633	2.826	NI
B.4	int-ext	706	2.427	99.513
B.5	protocol	714	2.447	101.389
B.6	dstpkts	662	2.695	NI
B.7	dstbytes	655	2.745	NI
B.8	srcpkts	661	2.669	NI
B.9	srcbytes	667	2.651	NI
B.10	duration	651	2.737	NI
B.11	dstport	694	2.539	101.547
B.12	srcport	705	2.472	100.888
B.13	dstip	667	2.685	NI
B.14	srcip	562	3.101	NI

Table 6.8. Results from the second round of reverse feature selection where a single feature is removed from feature set *B.4*; no feature set improved upon the results of feature set *B.4*

Set #	Features Removed	# Clusters	Entropy	Performance (%)
B.4	int-ext	706	2.427	100.000
B.15	int-ext, hourofday	475	3.443	NI
B.16	int-ext, res-eph	673	2.633	NI
B.17	int-ext, protocol	748	2.308	101.534
B.18	int-ext, dstpkts	717	2.419	101.235
B.19	int-ext, dstbytes	711	2.454	NI
B.20	int-ext, srcpkts	721	2.421	101.903
B.21	int-ext, srcbytes	711	2.444	NI
B.22	int-ext, duration	690	2.535	NI
B.23	int-ext, dstport	741	2.337	101.626
B.24	int-ext, srcport	737	2.319	100.368
B.25	int-ext, dstip	713	2.472	NI
B.26	int-ext, srcip	573	3.055	NI

Table 6.9. Results from forward feature selection; the performance from feature set *B.4* could not be improved as adding either feature did not lower the entropy

Set #	Feature Removed	Feature Added	# Clusters	Entropy	Performance (%)
B.4	int-ext	N/A	706	2.427	100.000
B.27	int-ext	byterate	530	3.055	NI
B.28	int-ext	logtypes	464	3.616	NI

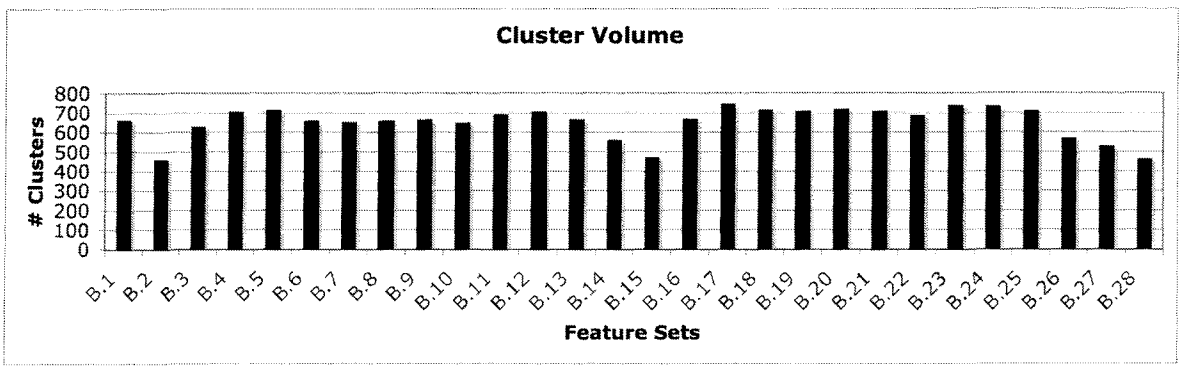

Figure 6.9. Cluster volume for expanded feature selection

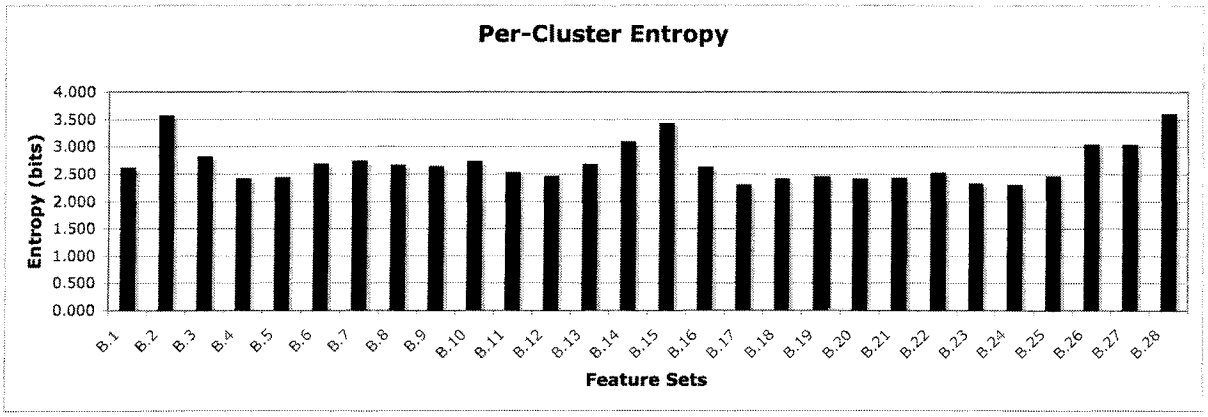

Figure 6.10. Per-cluster entropy for expanded feature selection

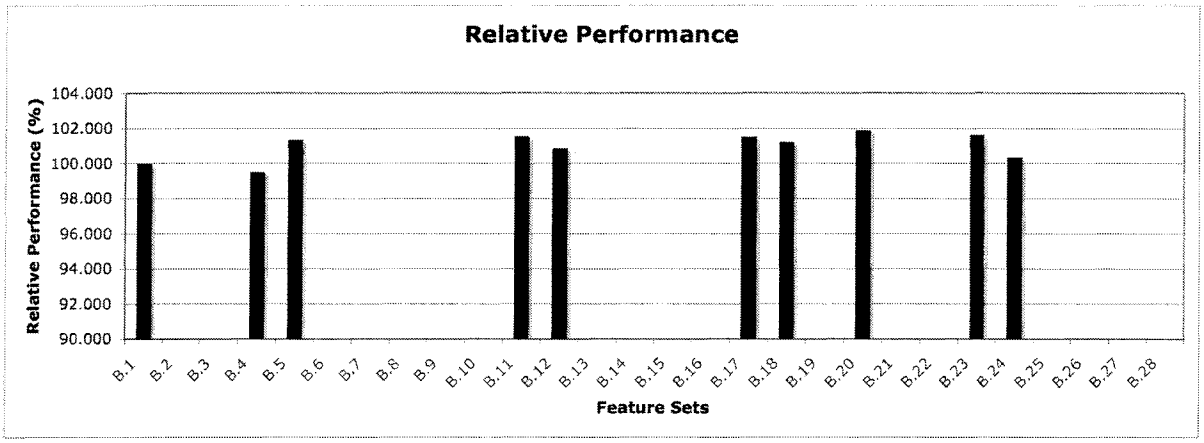

Figure 6.11. Relative performance for expanded feature selection; feature sets with empty values did not decrease the per-cluster entropy and therefore did not have a computed performance value

6.4.3. Reduced feature set results

The reduced feature set feature selection results are presented in tables 6.10, 6.11 and 6.12 and figures 6.12, 6.13 and 6.14. For this feature set, the only improvement over the initial feature set is obtained by removing the *dstport* feature during the first iteration of reverse feature selection. Interestingly, CV and PCE are improved by removing this feature. There are three fewer clusters (270 compared to 273) and the entropy decreased by 0.037 bits. Although not large improvements, this shows that the feature can be removed to reduce the number of features and improve correlation performance for both evaluation criteria.

Table 6.10. Results from the first iteration of reverse feature selection where a single feature is removed from the reduced feature set

Set #	Feature Removed	# Clusters	Entropy	Performance (%)
C.1	N/A	273	6.338	100.000
C.2	byterate	331	5.442	108.514
C.3	protocol	302	6.052	106.553
C.4	duration	229	7.409	NI
C.5	dstport	270	6.301	98.363
C.6	srcport	285	6.269	103.415
C.7	dstip	242	7.212	NI
C.8	Srcip	153	8.740	NI

Table 6.11. Results from the second round of reverse feature selection where a single feature is removed from feature set *C.5*; no feature set improved upon the results of feature set *C.5*

Set #	Features Removed	# Clusters	Entropy	Performance (%)
C.5	dstport	270	6.301	100.000
C.9	dstport, byterate	344	5.213	111.867
C.10	dstport, protocol	306	7.454	106.870
C.11	dstport, duration	220	6.180	NI
C.12	dstport, srcport	285	7.362	103.830
C.13	dstport, dstip	236	7.362	NI
C.14	dstport, srcip	138	8.867	NI

Table 6.12. Results from forward feature selection; the performance from feature set *C.5* could not be improved

Set #	Feature Removed	Feature Added	# Clusters	Entropy	Performance (%)
C.5	dstport	N/A	270	6.301	100.00
C.15	dstport	hourofday	513	3.973	156.750
C.16	dstport	logtypes	268	6.513	NI
C.17	dstport	res-eph	304	5.832	105.903
C.18	dstport	int-ext	258	6.753	NI
C.19	dstport	dstpkts	336	5.188	108.553
C.20	dstport	dstbytes	452	3.596	128.779
C.21	dstport	srcpkts	333	5.257	108.427
C.22	dstport	srcbytes	370	4.809	115.733

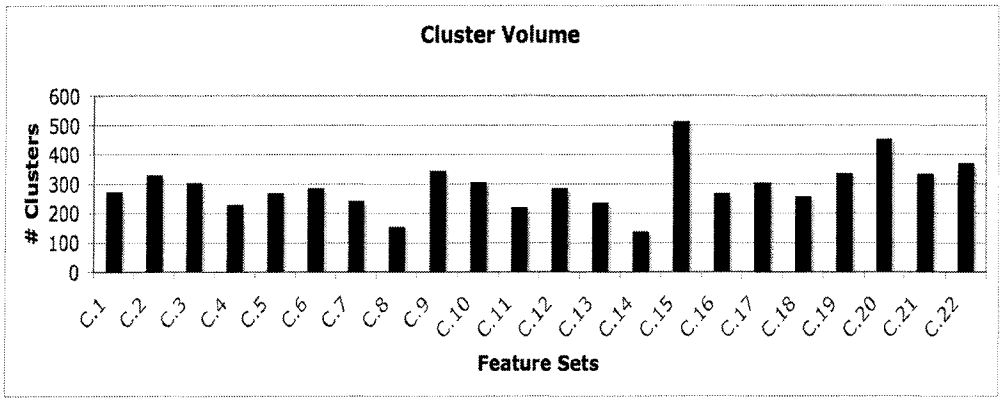

Figure 6.12. CV for reduced feature selection

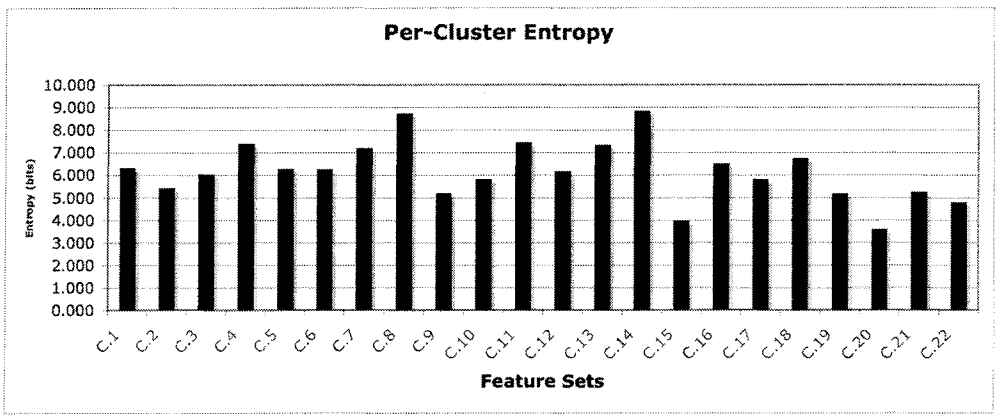

Figure 6.13. PCE for reduced feature selection

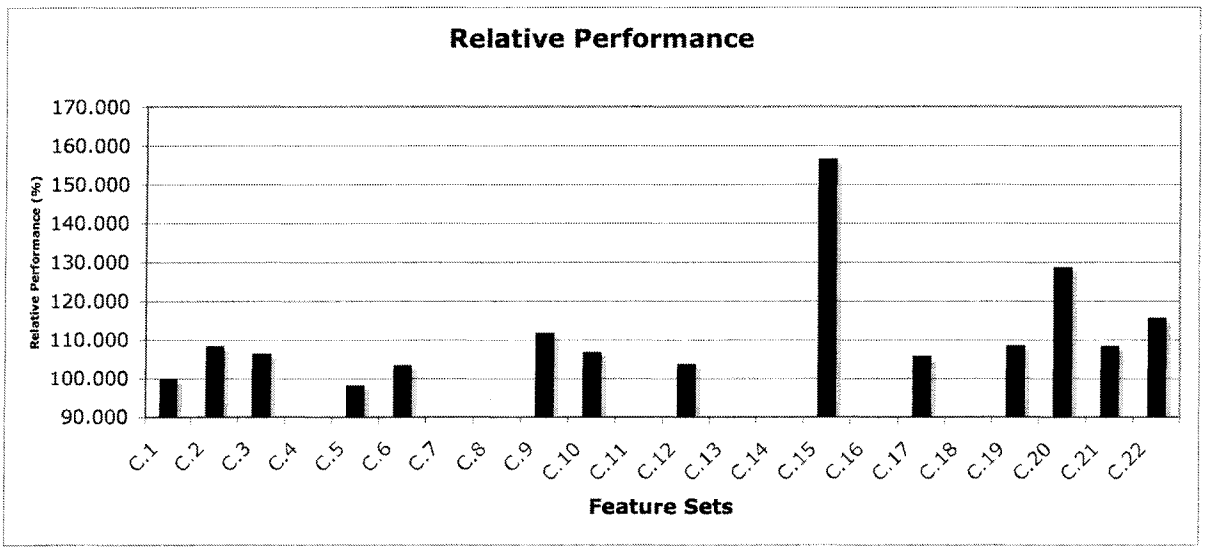

Figure 6.14. Relative performance for reduced feature selection; feature sets with empty values did not decrease the per-cluster entropy and therefore did not have a computed performance value

6.5. Summary

Ideally, feature selection for alert correlation uses labeled data to optimize a set of features by considering all possible feature combinations. This is unrealistic in production environments where labeled datasets of significant size are rarely available and the number of possible feature combinations is extremely large.

Our approach to feature selection allows feature sets to be compared on any number of weighted evaluation criteria, making feature selection possible on unlabeled datasets. Also, we use hill-climbing approaches to reverse and forward feature selection that locally optimize a candidate feature set based on the user-defined evaluation criteria. These two contributions make our approach ideal for production environments where labeled data is unavailable and feature selection must be run periodically on live data to tune alert correlation algorithms.

We have implemented our feature selection algorithm and applied it to our multi-paradigm alert correlation feature set. The results show that the performance of our candidate feature set can be improved by removing a single feature in terms of PCE and CV. Additionally, we test our implementation on alternate feature sets to show the feature selection process on reduced and expanded feature sets.

7. Conclusions and Future Work

7.1. Discussion

This dissertation addresses techniques useful for adapting IDS research to large-scale production networks. The end goal of our work is a new approach to alert correlation that is IDS independent. This is an important contribution, as current techniques require normalization and expert rules to perform correlation, which is difficult in practice. Our alert correlation approach is designed for multi-sensor environments where the number of alerts generated by IDSs outpaces the analysts' ability to evaluate alerts.

The correlation algorithm can be tuned on live datasets using our domain expert evaluation approach along with our hill climbing feature selection algorithm. This is an important aspect of our work, as current alert correlation techniques do not provide approaches or suggestions on how to tune alert correlation algorithms on production networks where labeled datasets are unavailable. Without the ability to tune and test correlation performance, alert correlation cannot be adequately integrated into the real-world toolset used by security analysts to perform their job duties. This evaluation approach can be applied to other alert correlation techniques besides our prototype system.

Our prototype alert correlation system is built upon a foundation of important tools: the data management and access framework and the alert verification and event correlation framework. The data management and access framework provides scalable data capture and a single view of the data via a web service. The alert verification and event correlation framework provides the ability to verify alerts and gather correlated events via software agents. Both frameworks are IDS and data independent and are easily adapted to a variety of network environments.

While these frameworks provide the foundation for building the alert correlation prototype, they address limitations of many research efforts in IDSs and can be extended to provide scalable data capture and access and automate repetitive tasks to fill a variety of IDS needs. Until now, generalized approaches to the areas addressed by each framework have not been designed for IDS research. Our work addresses this limitation and can be used by IDS

researchers to make their work more practical and applicable to production network environments.

7.2. Conclusions

This dissertation has presented a variety of solutions to problems facing the application of IDS research to large, production networks. The contributions of this work are important to developers and researchers in this area as the techniques developed here allow IDS research results to be deployed and tested in practice and used to provide security analysts with tools to better perform their jobs.

We first addressed limitation in scalability and usability inherent in managing and accessing large, heterogeneous datasets typically used in network security. We have presented a web service framework and implementation that provides uniform access to multi-terabyte datasets with minimal overhead. This system is currently deployed at LANL and provides analysts with access to billions of records from a variety of tools over a span of five years.

Next, we developed and implemented a framework for alert verification and event correlation. By using an agent-based approach, our system can be easily extended to perform a variety of functions, of which we presented several example agent implementations.

Building upon our alert verification and event correlation framework, we developed an IDS independent, multi-paradigm alert correlation algorithm. Our results were obtained from six months of network data generated at LANL. Domain expert security analysts evaluated our results, showing how alert correlation algorithms can be tested and tuned in real-world environments. Overall, our method of correlation was preferred to a technique correlating alerts using features in the Snort alert schema.

Finally, we developed a feature selection algorithm designed to run on production datasets to tune alert correlation results using user-defined performance metrics. We defined two metrics for performing feature selection on our alert correlation results: cluster entropy and cluster volume. Using the tradeoff in performance between the two measures we were able to reduce our feature set, improving performance in terms of the performance metrics and execution time.

As we have provided solutions to problems common to many IDSs, we feel that the existing body of research in this area can be more easily adapted for use in practice. Our hope is that both research, open source and commercial IDS developers will apply and expand upon the ideas and solutions presented in this dissertation to create a new class of scalable, flexible and accurate IDS suitable for deployment in large, production networks.

7.3. Future Work

To continue our work in extending intrusion detection research to production environments we hope to continue development of each of the four areas we have addressed in this dissertation: data management and access, alert information quality, alert correlation and performance evaluation. Specifics for each area are provided in the following subsections. Section 7.3.1 addresses our future work in web services and global-scale intrusion detection. Section 7.3.2 describes future development that we will explore to extend the alert verification and event correlation framework. Section 7.3.3 discusses our plan to improve upon multi-paradigm alert correlation. Section 7.3.4 concludes by addressing our plans to extend the feature selection and performance evaluation techniques that we have developed.

7.3.1. Data management and access

The results of the web service prototype have been encouraging and we hope to build on top of the web service by exploring the advantages of multi-site data sharing. With the system in place, it is now possible to explore emerging research areas including correlation of network events and detection of widely distributed network attacks. The growing number of XML data processing tools can now be applied to network security data sources. We hope to explore emerging XML technologies such as XSLT, XPath and XQuery to assist in knowledge acquisition, data mining and user interface development.

Future revisions of DiSARM's web service will be designed and implemented using web service standards such as SOAP, WSDL and UDDI. This will reduce the burden on developers and also allow for greater extensibility of the web service's functionality. As the

system is largely XML-based already, extending the model to incorporate web service standards will not be difficult. Also, current research such as Extended Service Oriented Architecture (ESOA) aims to address the problems of distributing web services between multiple sites [4]. This technology can be integrated into the web service framework to make multi-site data sharing easier to deploy for organizations like the United States' Department of Energy (DOE) or Department of Defense (DOD).

Also, privacy issues must be explored to motivate sites to deploy this web service. The sensitivity of network security data is an important issue and cannot be ignored. It is unlikely that sites will allow universal access to their data. Thus the system must provide mechanisms for protecting data. The access controls and authentication integrated into the system are a solid foundation but additional functionality such as integrated data obfuscation need to be explored.

7.3.2. Alert verification and event correlation

In continuing this research, we plan to focus on the development of novel alert verification agents. Also we would like to implement more complex agent dependencies to improve upon the hierarchical processing capabilities of the prototype system.

Finally we hope to make agent development easier by automating various aspects of creating and adding agents to the framework. This will involve moving from an HTTP based implementation to one using web service technology as we plan to do with the web service framework. Adopting web service standards in these two areas will be examined simultaneously.

7.3.3. Alert correlation

To improve upon our alert correlation results we plan to apply more advanced data mining techniques to our correlation model. The similarity measure and clustering algorithm are the two areas in which our results can be improved by using novel data mining techniques. Also, as new input datasets become available we will explore the impact of the data on the correlation process. Most importantly, we will begin extending the correlation

algorithm to incorporate additional alert datasets beyond Snort. We plan to show that our technique is applicable to IPS and ADS alerts as well as misuse detection systems.

As our current prototype implementation is an off-line post-processing system, we plan to implement a new real-time, online system based on the prior implementation. This will involve creating real-time event correlation agents to assist in the collection of log data for alert profile generation. Also, the clustering algorithm needs to be rewritten to provide the ability to add alerts to clusters without rebuilding the entire dataset.

7.3.4. Feature selection and performance evaluation

We plan to expand the number of evaluation criteria and determine additional candidate feature sets. Also we hope to explore randomized versions of our feature selection algorithm to evaluate feature sets that were not considered in our current implementation.

Tuning of our correlation results will require additional domain expert evaluations, which require a large time commitment from the analysts. We hope to examine means of streamlining this process to optimize the effectiveness of the evaluation while making the process easier on the analysts. One approach is to develop user feedback capabilities in the software prototypes to allow analysts to evaluate results during their daily operations.

APPENDIX A. Query Document Type Definition

This appendix presents the generalized query schema discussed in chapter 3. To extend the query schema, additional attributes can be added to the *Query* element. The attributes shown in italic, *srcip*, *dstip* and *dstport*, are currently supported by the LANL implementation. Additional fields such as MAC address or source port can be added to the schema to fit the needs of the site deploying the web service.

```
<?xml version="1.0"?>
<!ELEMENT Query (Query*, Timestamp*, DataType*, SearchField*)>
<!ELEMENT Timestamp EMPTY>
<!ELEMENT Datatype EMPTY>
<!ELEMENT SearchField EMPTY>
<!ATTLIST Query srcip CDATA>
<!ATTLIST Query dstip CDATA>
<!ATTLIST Query dstport CDATA>
<!ATTLIST Query user CDATA>
<!ATTLIST Query password CDATA>
<!ATTLIST Timestamp start CDATA #REQUIRED >
<!ATTLIST Timestamp end CDATA>
<!ATTLIST SearchField name CDATA #REQUIRED>
<!ATTLIST SearchField format CDATA>
<!ATTLIST SearchField predicate (=|%gt;|%lt;|%gt;=|%lt;=|~) #REQUIRED>
<!ATTLIST SearchField value CDATA>
<!ATTLIST SearchField lower CDATA>
<!ATTLIST SearchField upper CDATA>
<!ATTLIST DataType type CDATA #REQUIRED>
<!ATTLIST DataType sensor CDATA>
```

APPENDIX B. Result Document Type Definition

The generalized result schema discussed in chapter 3 is presented in this appendix. To extend this schema, elements are added to the *QueryResult* element, which will contain entries for each data type being made available to query via the web service. As an example, the XML DTD additions for LANL's LFAP data type are included and shown in italics. LFAP data records are unidirectional flow records generated by cflowd, a network connection generation process running on LANL's internal routers.

```
<?xml version="1.0"?>
<!ELEMENT Result (QueryError*, QueryResult*, DataType*)>
<!ELEMENT QueryError EMPTY>
<!ELEMENT QueryResult (LFAP*, type1*, type2*, ... , typen*)>
<!ELEMENT DataType (Sensor+, SearchField+)>
<!ELEMENT Sensor EMPTY>
<!ELEMENT SearchField (Predicate+, Format+)>
<!ELEMENT Predicate (EMPTY)>
<!ELEMENT Format (EMPTY)>
<!ELEMENT LFAP (EMPTY)>
<!ATTLIST Result incident CDATA>
<!ATTLIST DataType id CDATA #REQUIRED >
<!ATTLIST DataType name CDATA>
<!ATTLIST Sensor id CDATA #REQUIRED >
<!ATTLIST Sensor name CDATA>
<!ATTLIST SearchField name CDATA #REQUIRED >
<!ATTLIST SearchField description CDATA #REQUIRED >
<!ATTLIST Predicate value (=|%gt;|%lt;|%gt;=|%lt;=|~) #REQUIRED>
<!ATTLIST Format name CDATA #REQUIRED>
<!ATTLIST Format description CDATA>
<!ATTLIST Format default CDATA>
<!ATTLIST LFAP sensor CDATA #REQUIRED>
<!ATTLIST LFAP srcip CDATA #REQUIRED>
<!ATTLIST LFAP srcport CDATA #REQUIRED>
<!ATTLIST LFAP dstip CDATA #REQUIRED>
<!ATTLIST LFAP dstport CDATA #REQUIRED>
<!ATTLIST LFAP bytes CDATA #REQUIRED>
<!ATTLIST LFAP pkts CDATA #REQUIRED>
<!ATTLIST LFAP startts CDATA #REQUIRED>
<!ATTLIST LFAP endts CDATA #REQUIRED>
<!ATTLIST LFAP protocol CDATA #REQUIRED>
<!ATTLIST LFAP router CDATA #REQUIRED>
```

APPENDIX C. Alert Database Relation Schemas

This appendix details the relations used in our prototype alert verification and event correlation framework implementation presented in chapter 4. Table C.1 shows the relations, their schemas and a brief description. Section 4.3 provides additional detail on the usage of these relations in the prototype. The schemas for the type-specific alert relations are omitted. Three such relations are used in our prototype, one for each of the supported IDSs: Snort, TippingPoint and EMAAD.

Table C.1. List of relations used in framework implementation.

Relation	Schema	Description
alert state	< Alert ID, Agent ID, Evaluation, State >	Mapping of alert state and evaluation to agents
universal alert	< Alert ID, Agent ID, Timestamp >	Primary table for storing alert information
type-specific alert	< Alert ID, fields for IDS alert... >	Per-IDS table for IDS-specific fields
dependencies	< Dependent Agent ID, Parent Agent ID, Dependency >	Mapping of agent dependencies
information content	< Alert ID, Agent ID, content >	Free-text information content from evaluation agent
correlated events	< Alert ID, Agent ID, event >	Correlated events from event correlation agent stored as XML

APPENDIX D. Feature Vector

Table D.1 described the feature vector schema used in the evaluation of our multi-paradigm alert correlation algorithm in chapter 5. Selecting features and histogram ranges is a highly subjective process that is dependent on domain knowledge. Most features described are taken directly from fields in network traffic and intrusion detection log records. The *Network Relation* and *Port Relation* features are abstract features.

Table D.1. Feature vector abbreviations and histogram range descriptions

Field	Abbreviation	Histogram Values	Description
Source IP	srcip	[0 ... 255]	the class A network of the IP
Destination IP	dstip	[0 ... 255]	the class A network of the IP
Source Port	srcport	[X, Y]	X < 1024, Y >= 1024
Destination Port	dstport	[X, Y]	X < 1024, Y >= 1024
Duration	duration	[0 ... 9]	non-linear range of integer values
Source Bytes	srcbytes	[0 ... 9]	non-linear range of integer values
Source Packets	srcpkts	[0 ... 9]	non-linear range of integer values
Dest. Bytes	dstbytes	[0 ... 9]	non-linear range of integer values
Dest. Packets	dstpkts	[0 ... 9]	non-linear range of integer values
Protocol	protocol	[TCP, UDP, ICMP]	
Network Relation	int-ext	[X→Z,Y→Z, Z→X,Z→Y]	X and Y are internal networks, Z is all external networks
Port Relation	res-eph	[X→Y,Y→X, X→X,Y→Y]	X < 1024, Y >= 1024

The *Source IP* and *Destination IP* address features are divided into 256 bins each and the bin value is incremented according to the upper eight bits of the IP address. The *Source Port* and *Destination Port* features are divided into two bins each, one for reserved ports under 1024 and the other for ephemeral ports 1024 and above. The *Duration*, packet and byte features are divided into bins according to the ranges in table D.2. The bin count is only incremented for the first bin the matches the record's value for that field. The *Protocol* feature defines bins for TCP, UDP and ICMP and increments the count for each record according to its protocol. The *Network Relation* feature tracks the relationship between the source IP and the destination IP in the record. Internal networks are defined as X and Y with

Z being all external networks. For example if the source IP of a record is in Y and the destination address is in Z then the bin for Y→Z is incremented. The *Port Relation* feature is similar to the *Network Relation* feature, tracking the relation between source and destination port. For example, the X→X bin is incremented when a record contains reserved ports for both the source and destination ports.

Table D.2. Miscellaneous histogram ranges

Feature	Category Ranges
Duration (sec)	0, 1-5, 6-10, 11-60, 61-120, 121-300, 301-600, 601-1200, 1201-1800, 1801+
Source Bytes (100s bytes)	0-1, 2-10, 11-50, 51-100, 101-500, 501-1000, 1001-5000, 5001-10000, 10001-50000, 50001+
Source Packets	0-3, 4-7, 8-15, 16-30, 31-50, 51-100, 101-500, 501-1000, 1001-5000, 5001+
Destination Bytes (100s bytes)	0-1, 2-10, 11-50, 51-100, 101-500, 501-1000, 1001-5000, 5001-10000, 10001-50000, 50001+
Destination Packets	0-3, 4-7, 8-15, 16-30, 31-50, 51-100, 101-500, 501-1000, 1001-5000, 5001+

APPENDIX E. Entropy Calculations

$$entropy(c) = \sum_{i=1}^{N}\left(-1 * \sum_{j=1}^{Mi} p_j \log_2 p_j\right) \quad \textbf{(E.1)}$$

Equation E.1 calculates the entropy in bits for a cluster of Snort alerts c where N is the number of features. Mi is the number of distinct items for feature i and p_j is the probability that the j^{th} value of feature i is found in cluster C. For each cluster of Snort alerts, entropy is calculated and summed for five features: source and destination IP addresses, source and destination ports, and signature name. The function shown in figure E.1 is used to compute the average entropy for a set of clusters.

Input: C - a set of clusters of Snort alerts
Output: per-cluster entropy of C in bits

```
CalculateEntropy (C)
      e = 0.0
      count = size(C)
      for i = 1 to count
            e += entropy(C[i])
      e = e / count
      return e
```

Figure E.1. *CalculateEntropy* function used in prototype feature selection algorithm

APPENDIX F. Additional Candidate Feature Selection Results

This appendix presents additional results from the candidate feature selection process in chapter 6. Table 6.4 in section 6.4.1 shows that removing three features individually (*int-ext*, *protocol* and *srcport*) from the candidate feature set results in improved performance. The algorithm is recursive and runs reverse feature selection on each of these three feature sets. Section 6.4.1 only presents results from the *protocol* execution path, which ultimately results in the best performance of the three paths. For completeness, we show the second round of reverse feature selection for the *int-ext* and *srcport* paths. As shown in tables F.1 and F.2, neither of the feature sets can be further reduced, thus showing that feature set *A.4* is the optimal feature set in this case.

Table F.1. Results from the second iteration of reverse feature selection where the *int-ext* feature has been removed from the candidate feature set; feature set *A.3* is compared to feature sets *A.28* to *A.38*

Set #	Features Removed	# Clusters	Entropy	Performance (%)
A.3	int-ext	475	3.443	100.000
A.28	int-ext, res-eph	443	3.590	NI
A.29	int-ext, protocol	529	3.037	101.198
A.30	int-ext, dstpkts	479	3.538	102.962
A.31	int-ext, dstbytes	476	3.617	NI
A.32	int-ext, srcpkts	489	3.498	104.131
A.33	int-ext, srcbytes	470	3.559	101.534
A.34	int-ext, duration	467	3.536	100.305
A.35	int-ext, dstport	512	3.206	101.760
A.36	int-ext, srcport	527	3.100	102.345
A.37	int-ext, dstip	462	3.757	NI
A.38	int-ext, srcip	368	3.956	NI

Table F.2. Results from the second iteration of reverse feature selection where the *srcport* feature is removed from the candidate feature set; feature set *A.11* is compared to feature sets *A.39* to *A.49*

Set #	Features Removed	# Clusters	Entropy	Performance (%)
A.11	srcport	488	3.325	100.000
A.39	srcport, res-eph	445	3.644	NI
A.40	srcport, int-ext	527	3.100	102.345
A.41	srcport, protocol	536	2.988	101.483
A.42	srcport, dstpkts	477	3.489	101.291
A.43	srcport, dstbytes	481	3.524	103.049
A.44	srcport, srcpkts	483	3.427	101.031
A.45	srcport, srcbytes	475	3.556	102.542
A.46	srcport, duration	469	3.561	101.370
A.47	srcport, dstport	511	3.185	101.009
A.48	srcport, dstip	476	3.582	NI
A.49	srcport, srcip	378	3.867	-

Bibliography

[1] "Snort: The Open Source Network Intrusion Detection System". http://www.snort.org. Page visited July 10, 2006.

[2] Sekar, R., Guang, Y., Verma, S., Shanbhag, T. "A High-Performance Network Intrusion Detection System", *Proceedings of the 6th ACM Conference on Computer and Communications Security*, November 1999.

[3] Kruegel, C., Vigna, G., Valeur, F., Kemmerer, R. "Stateful Intursion Detection for High-Speed Networks", *Proceedings of the IEEE Symposium on Security and Policy*, November 2002.

[4] Somayaji, A., Hofmeyer, S. A., Forrest, S. "Principles of a Computer Immune System", *Proceedings of the New Computer Security Paradigms Workshop*, 1997.

[5] Kosoresow, A. P., Hofmeyer, S. A. "Intrusion Detection via System Call Traces", *IEEE Software*. September/October 1997.

[6] Maxion, R. A., Tan, K. M. C. "Anomaly Detection in Embedded Systems", *IEEE Transactions on Computers*, Vol. 51, No. 2, February 2002.

[7] Michael, C. C., Ghosh, A. "Simple, State-Based Approaches to Program-Based Anomaly Detection", *ACM Transactions on Information and System Security*, Vol. 5, No. 3, August 2002, Pages 203–237.

[8] R. Sekar, A. Gupta et al. "Specification-based anomaly detection: a new approach for detecting network intrusions", *ACM Computer and Communication Security Conference*. 2002.

[9] Ye, N., Vilbert, S., Chen, Q. "Computer Intrusion Detection Through EWMA for Autocorrelated and Uncorrelated Data", *IEEE Transactions on Reliability*. March 2003"

[10] Lane, T., Brodley, C. E. "Temporal Sequence Learning and Data Reduction for Anomaly Detection", *Proceedings of the 5th ACM Conference on Computer and Communications Security*, April 1998.

[11] Taylor, C., Alves-Foss, J. "NATE - Network Analysis of Anomalous Traffic Events, A Low-cost Approach", *Proceedings of the New Security Paradigms Workshop*, September 10-13 2001.

[12] Taylor, C., Alves-Foss, J. "An Emperical Analysis of NATE - Network Analysis of Anomalous Traffic Events", *Proceedings of the New Security Paradigms Workshop*, September 23-26 2002.

[13] Cuppens, F. "Managing Alerts in a Multi-Intrusion Detection Environment", *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, 2002.

[14] Curry, D., Debar, H., "Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition". http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-12.txt, 2004.

[15] Valdes, A., Skinner, K. "Probabilistic Alert Correlation", *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, 2001.

[16] Ning, P., Cui, Y., Reeves, D. S. "Constructing Attack Scenarios through Correlation of Intrusion Alerts", *Proceedings of the 9th ACM Conference on Computer and Communications Security*, November 2002.

[17] Cuppens, F., Miège, A. "Alert Correlation in a Cooperative Intrusion Detection Framework", *Proceedings of the IEEE Symposium on Security and Policy*, November 2002.

[18] Lee, W., Stofolo, S. J. "A Framework for Constructing Features and Models for Intrusion Detection Systems", *ACM Transactions on Information and System Security*, Vol. 3, No. 4, November 2000, pp. 227–261.

[19] Lee, W. "Applying Data Mining to Intrusion Detection: the Quest for Automation, Efficiency and Credibility", *SIGKDD Explorations*, 2002, Volume 4, Issue 2.

[20] Yang, X., Shen, J., Liu, Q. "A Novel Clustering Algorithm Based on Weighted Support and its Applications", *Proceedings of the First International Conference on Machine Learning and Cybernetics*, April 2002.

[21] Reilly, M., Stillman, M. "Open Infrastructure for Scalable Intrusion Detection", *Proceedings of the DARPA Information Survivability Conference and Exposition*, January 2000, pp. 25-27.

[22] Helmer, G., Wong, J. S. K., Honavar, V., Miller, L. "Intelligent agents for intrusion detection" *Proceedings of the IEEE Information Technology Conference*, September 1998, pp. 121-124.

[23] Ning, P., Jajodia, S., Xiaoyang, S. W. "Abstraction-based intrusion detection in distributed environments", *ACM Transactions on Information and System Security*, Vol. 4, No. 4, November 2001, pp. 407-452.

[24] Zissman, M., "DARPA Intrusion Detection Evaluation". http://www.ll.mit.edu/IST/ideval/, 2001. Page visited July 10, 2006.

[25] Bass, T. "Intrusion Detection Systems and Multisensor Data Fusion", *Communications of the ACM*, Vol. 43, No. 4, April 2000, pp. 99-105.

[26] Tung, B. "The Common Intrusion Specification Language: a retrospective", *Proceedings of the DARPA Information Survivability Conference and Exposition*, January 2000.

[27] Garcia-Molina, H., Ullman, J. D., Widom, J. Database Systems: The Complete Book. Prentice Hall, Upper Saddle River, NJ, pp 1053-1070, 2002.

[28] "Anti-Terrorism, Crime and Security Act". http://www.publications.parliament.uk/pa/cm200102/cmbills/049/2002049.pdf, 2002. Page visited July 10, 2006.

[29] Campione, M., et. al. The Java Tutorial Continued: The Rest of the JDK. Sun Microsystems, Palo Alto, CA, pp 359 – 394, 1999.

[30] Stevens, W. R. UNIX Network Programming, 2nd Edition. Prentice Hall, Upper Saddle River, NJ, pp 399 – 452. 1999.

[31] Graham, S., et. al. Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI. Sams Publishing. 2002.

[32] "Google SOAP Search API (beta)". http://www.google.com/apis/. Page visited July 10, 2006.

[33] Paolucci, M., et. al. Discovery of Information Sources across Organizational Boundaries. *Proceedings of the IEEE International Conference on Services Computing*, 2005.

[34] Danyliw, R., Meijer, J., Demchenko Y., "The Incident Object Description Exchange Format Data Model and XML Implementation". http://www.ietf.org/internet-drafts/draft-ietf-inch-iodef-04.txt, 2005. Page visited September 2005.

[35] Valeur, F., Vigna, G., Kruegel, C., Kemmerer, R. A., "A Comprehensive Approach to Intrusion Detection Alert Correlation", *IEEE Transactions on Dependable and Secure Computing*. Vol 1, No 3, pp. 146-169. July-September 2004.

[36] Gauch, H. G. Jr., "A Quantitative Evaluation of the Bray-Curtis Ordination", *Ecology*, Vol 54, No 4, pp. 829–836. 1973.

[37] Fung, G., Mangasarian, O. L. "Proximal support vector machine classifiers". *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

[38] Portnoy, L., Eskin, E., Stolfo, S. "Intrusion detection with unlabeled data using clustering". *Proceedings of the ACM Workshop on Data Mining Applied to Security*, 2001.

[39] Zhicai, S., Zhenzhou, J., Mingzeng, H., "A novel distributed intrusion detection model based on mobile agent", *Proceedings of the 3rd international Conference on information Security,* November 14 - 16, 2004.

[40] Huang, Y, Lee, W., "A cooperative intrusion detection system for ad hoc networks", *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks,* 2003.

[41] Foukia, N., "IDReAM: intrusion detection and response executed with agent mobility architecture and implementation", *Proceedings of the Fourth international Joint Conference on Autonomous Agents and Multiagent Systems,* 2005.

[42] Ning, P., Jajodia, S., Wang, X. S., "Abstraction-based intrusion detection in distributed environments", *ACM Transactions on Information Systems Security.* Vol 4, No 4, pp 407-452, November 2001.

[43] Vigna, G., Valeur, F., Kemmerer, R. A., "Designing and implementing a family of intrusion detection systems", *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT international Symposium on Foundations of Software Engineering,* Helsinki, Finland, September 1-5, 2003.

[44] Lee, W., Stolfo, S. J., "A framework for constructing features and models for intrusion detection systems", *ACM Transactions on Information Systems Security,* Vol 3, No 4, pp 227-261, November 2000.

[45] Levchenko, K., Paturi, R., Varghese, G., "On the difficulty of scalably detecting network attacks", *Proceedings of the 11th ACM Conference on Computer and Communications Security,* Washington, D.C., October 25-29, 2004.

[46] Julisch, K., "Clustering intrusion detection alarms to support root cause analysis", *ACM Transactions on Information Systems Security,* Vol 6, No 4, pp 443-471, November 2003.

[47] Sekar, R., et al, "Specification-based anomaly detection: a new approach for detecting network intrusions", *Proceedings of the 9th ACM Conference on Computer and Communications Security,* Washington, D.C., November 18-22, 2002.

[48] Lee, W., Stolfo, S. J., Mok, K. W., "Mining in a data-flow environment: experience in network intrusion detection", *Proceedings of the Fifth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*, San Diego, California, August 15 - 18, 1999.

[49] Kruegel, C., Vigna, G. "Anomaly detection of web-based attacks", *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, D.C., October 27-30, 2003.

[50] Han, H., et al, "Data mining aided signature discovery in network-based intrusion detection system", *SIGOPS Operating Systems Review,* Vol 36, No 4, October 2002.

[51] Viinikka, J., et al, "Time series modeling for IDS alert management", *Proceedings of the 2006 ACM Symposium on information, Computer and Communications Security,* Taipei, Taiwan, March 21-24, 2006.

[52] Guangchun, L., et al, "MADIDS: a novel distributed IDS based on mobile agent", *SIGOPS Operating Systems Review*, Vol 37, No 1, pp 46-53, January 2003.